

سلسلة ملخصات شوم  
نظريات ومسائل في

البرمجة بلغة

# الباسكال

بايرون س، جوتفريد

أفضل وسيلة مساعدة للطالب تجعله متميزاً في الإختبارات ويحصل  
على أعلى الدرجات :

- ٣٩١ مسألة محلولة حلاً كاملاً
- مئات من المشاكل المتكاملة
- ٣٥ مثال لبرامج كاملة
- يستخدم مع كل الكتب الدراسية الرئيسية
- يقدم عمق شامل في حل المشاكل



الدار الدولية للنشر والتوزيع





**البرمجة بلغة الباسكال**





ملخصات شوم

نظريات ومسائل

فنى

# البرمجة بلغة الباسكال

تأليف

بايرون س. جوتفريد Ph. D.

أستاذ الهندسة الصناعية

هندسة إدارة النظم وبحوث العمليات

جامعة بيتسبرج

ترجمة ومراجعة

دكتور/ سرور على إبراهيم سرور

كلية الاقتصاد والإدارة

جامعة الملك سعود - فرع القصيم

تقديم

دكتور / عبد الله بن عبد الله العبيد

عميد كلية الاقتصاد والإدارة

جامعة الملك سعود - فرع القصيم

الدار الدولية للنشر والتوزيع



## حقوق النشر

الطبعة الإنجليزية : حقوق التأليف © ١٩٨٥ ، دار ماكجروهيل للنشر ،

جميع الحقوق محفوظة.

Theory and problems of  
PROGRAMMING WITH PASCAL

by

BYRON S. GOTTFRIED

الطبعة العربية الأولى : حقوق الطبع والنشر © ١٩٩١ ، جميع الحقوق محفوظة للناشر :

**الدار الدولية للنشر والتوزيع**

ص.ب ٥٥٩٩ هليوبوليس غرب - القاهرة

ت : ٢٥٨٢٨٨٧

تلكس : ٢٠٠٧٠ UN PBCRB

فاكس : ٢٩١٨٠٥٩ / ٢٠٢٠٢

لا يجوز نشر أى جزء من هذا الكتاب أو اختزان مادته بطريقة الاسترجاع أو نقله على أى وجه أو بأى طريقة سواء كانت اليكترونيه أو ميكانيكيه أو بالتصوير أو بالتسجيل أو خلاف ذلك الا بموافقة الناشر على هذا كتابة ومقداً.

بسم الله الرحمن الرحيم

## تقديم

الحمد لله رب العالمين ، والصلاة والسلام على أشرف المرسلين ، سيدنا محمد ، وعلى آله وصحبه ، ومن تبعهم بإحسان إلى يوم الدين .

وبعد :

فما من شك في أن تطوير القدرات والمهارات لاستخدام جهاز الكمبيوتر أصبح من الأمور الضرورية في هذا العصر . فهذا الجهاز توغل في حياتنا ، حتى أصبح مؤثراً تأثيراً مباشراً أو غير مباشر على كل فرد منا .

فمما هو جدير بالذكر أن هناك لغات برمجة خاصة تستخدم مع الكمبيوتر . ومن أهم هذه اللغات في وقتنا الحالي « لغة البسكال » التي سميت باسم عالم الرياضيات والفيزياء الفرنسي « بلز بسكال » تكريماً له ، حيث إنه أعد عام ١٦٤٥ م آلة حاسبة قادرة على جمع وطرح الأعداد . وقد كان لإعداد هذه الآلة دور كبير في الأوساط العلمية التي لم تكن تتوقعها ، مما جعل الناس لا يتقبلونها بيسر في ذلك الوقت . وبعد ظهور الكمبيوتر وانتشاره ؛ لاقت لغة البسكال انتشاراً سريعاً ، حيث إنها أعدت لعمل برمجة مرتبة ومنظمة للكمبيوتر .

وقد وقع الاختيار على كتاب « البرمجة بلغة البسكال » لمؤلفه الدكتور / بيرون جوتفريد ، أستاذ الهندسة الصناعية بجامعة بتسبرج بأمريكا ، لأهمية المادة العلمية الموجودة فيه ، وسهولة عرضه لها . وحتى تتاح الفرصة للاستفادة من هذا الكتاب على وجه أفضل ، اتجه الرأي إلى ترجمته ليستخدم كأساس للتدريس في الدورات التدريبية التي يقدمها مركز البحوث وتنمية الموارد البشرية بكلية الاقتصاد والإدارة . وقام بترجمته الدكتور / سرور على إبراهيم سرور ، الأستاذ المشارك بقسم الأساليب الكمية بالكلية ، وقام بمراجعته الأستاذ / عاصم أحمد الصاحمي ، المشرف على معمل الكمبيوتر بالكلية .

هذا .. ونرجو أن يثيبها الله خير الثواب على ما قاما به من جهد مشكور في إعداد هذا الكتاب . كما نأمل أن يكونا قد قدما إلى مكتبتنا العربية كتاباً جديداً في بابها ، يحقق الغرض المرجو ، ويحقق الهدف المقصود .

والله من وراء القصد ،

د . عبد الله بن عبد الله العبيد

عميد كلية الاقتصاد والإدارة

جامعة الملك سعود - فرع القصيم



بسم الله الرحمن الرحيم

## مقدمة المترجم

أن أساس لغة البسكال هي لغة الجول ٦٠ ( Algol 60 ) . وقد أعدت لغة الجول ٦٠ في أصلها كلغة للتفاهم بين علماء الرياضيات . ومع وجود الكمبيوتر ، وجد أنه من الممكن استخدام هذه اللغة كأحدى لغات البرمجة للكمبيوتر . وهذا يبين لنا لماذا لم تلق هذه اللغة نجاحا كبيرا كأحدى لغات البرمجة .. فهي لغة صممت أساسا للتفاهم بين علماء الرياضيات . أما البسكال - والتي بنيت أساسا على لغة الجول ٦٠ - فهي لغة برمجة أعدت لعمل برمجة مرتبة ومنظمة . وقد يكون هذا هو السبب الرئيسي في انتشارها السريع ؛ فلم تلق أى لغة من لغات البرمجة مثل هذه السرعة في استخدامها .

ويتوقع أن يستمر استخدام لغة البسكال استخداما كبيرا مع جميع أجهزة الكمبيوتر خلال الخمسة وعشرين عاما المقبلة بإذن الله تعالى . ونظراً لأهمية هذه اللغة ، فقد تمت ترجمة هذا الكتاب لتوفير كتاب باللغة العربية في هذه اللغة الهامة للقارئ العربي .

ولايفوتني أن أقدم خالص شكرى لكل من ساهم في هذا العمل لإخراجه فى صورته الحالية ، وأخص بالذكر سعادة الدكتور / عبد الله بن عبد الله العبيد ، عميد كلية الاقتصاد والإدارة لما يوليه من اهتمام كبير لترجمة أمهات الكتب ، وتقديم كل ما يمكن تقديمه من جديد في العلم للقارئ العربي .

وعلى الله قصد السبيل .

المترجم





## مقدمة الكتاب

لقد ظهرت لغة البسكال كالعاصفة في العالم ، منذ وجودها في بداية السبعينيات . ففي خلال عدّة سنوات بسيطة ، أصبحت اللغة لغة قياسية لطلبة البرمجة الجادين في معظم الكليات والجامعات . وتدرس هذه اللغة الآن في العديد من المدارس الثانوية ، وفي الكليات . كما تحوّل العديد من المشتغلين بالكمبيوتر أيضا إلى البسكال ، مفضلين هذه اللغة عن لغات أخرى يسهل الوصول إليها . ومن الواضح أنه لم توجد لغة برمجة أخرى أثّرت هذا التأثير الكبير على مجتمع الكمبيوتر في فترة زمنية وجيزة ، مثلما ما فعلت لغة البسكال .

ولماذا كل هذا الاهتمام بلغة البسكال . هناك العديد من الأسباب ... أولا لأن لغة البسكال لغة منظّمة . ففي واقع الأمر .. لقد صمّمت لغة البسكال لترويج منهج منظّم ومنسّق لبرمجة الكمبيوتر . ويشجّع استخدامها تطوير البرامج المنظّمة تنظيما جيدا ، والمكتوبة بوضوح ، والخالية نسبيا من الخطأ سواء استخدمت مع أجهزة الكمبيوتر الكبيرة ، أم الصغيرة ، أم أجهزة الميكروكمبيوتر رخيصة الثمن . ويمكن استخدام اللغة في أي وسط برمجة بكفاءة مرتفعة ، سواء أكان وسط البرمجة هو تشغيل الدقعة ، أم التشغيل المتداخل . وعلى هذا ... فتقدّم لغة البسكال عددا من الخواص الجيدة غير المسيرة نسبيا لبعض لغات البرمجة الأخرى .

ويقدّم هذا الكتاب تعليمات برمجة الكمبيوتر ، مستخدما معال لغة البسكال القياسية (ISO) . والكتاب استمرارا لكتب البرمجة الأخرى في مجموعة سلسلة كتب شوم . وعلى هذا .. يقدّم الكتاب توضيحات كاملة ومفهومة لكل المعالم الرئيسية للغة البسكال ، مدعّمة بعدد كبير من الأمثلة التوضيحية . بالإضافة إلى ذلك ... يقدّم الكتاب منهجا معاصرا للبرمجة ، مع التركيز على أهمية الوضوح والكفاءة في إعداد البرنامج . وعلى هذا ... يقدّم للقارئ مبادئ الخبرة في برمجة جيدة ، مع تقديم القواعد الخاصة بلغة البسكال .

وقد روعي أن يكون أسلوب الكتابة بسيطا ، حتى يمكن أن يستخدمه الكثير من القراء من طلبة المدارس الثانوية ، وحتى المشتغلين بالكمبيوتر نوى الخبرة . ويناسب الكتاب - بصفة خاصة - طلبة أعلى مستوى في المدارس الثانوية ، وأول مستوى في الجامعات ، ككتاب مدرسي مقرّر في البرمجة ، أو ككتاب مساعد ، أو كدليل فعّال للدراسة الذاتية .

وفي معظم أجزاء الكتاب ، لايزيد مستوى الرياضيات المطلوب عن الجبر الذي يدرس في المدارس الثانوية . ومعظم المادة المقدّمة لا تتطلب خلفية رياضية في الواقع .

والمادة التي بالكتاب ، مقدّمة بطريقة تجعل القارئ قادرا على كتابة برامج بسكال كاملة وبسيطة بسرعة . ومن المهم جداً أن يكتب القارئ مثل هذه البرامج وينفّذها أثناء قراءته مادة الكتاب . وسوف يساعد ذلك المبرمجين المبتدئين على اكتساب الثقة بالنفس ، وجذب انتباههم للموضوع ، وذلك بصورة كبيرة . ( ويجب أن يكون مفهوما أن برمجة الكمبيوتر ماهي إلا مهارة ، مثل التأليف القصصي ، أو التأليف الموسيقي . ولا يمكن اكتساب مثل هذه المهارة بمجرد قراءة كتاب ) .

ويوجد عدد كبير من الأمثلة كجزء مكمل للكتاب . وتشمل هذه الأمثلة أمثلة برمجة متعددة شاملة ، كما تشمل مشاكل بسيطة تحتاج إلى تفكير . كما أن هناك تمارين برمجة إضافية موضحة في المشاكل المحلولة ، وتظهر في نهاية معظم الفصول . هذه الأمثلة والمشاكل المحلولة يجب دراستها بعناية مع استمرار القارئ في قراءة فصول الكتاب ، وبدئه كتابة برامج خاصة .

كما يوجد في نهاية كل فصل مجموعة من أسئلة المراجعة ، والمشاكل المتكاملة ، ومشاكل البرمجة . وتمكن أسئلة المراجعة القارئ من اختبار قدرته على تذكر المادة المقدمة في كل فصل . كما تقدم أيضا ملخصا فعالا للفصل . ولا تتطلب معظم المشاكل المتكاملة ومشاكل البرمجة خلفية رياضية أو تقنية خاصة . ويجب أن يحل الطالب كل ما يستطيع حله من هذه المشاكل . ( وتوجد في نهاية الكتاب إجابة للعديد من هذه المشاكل ) . وعند استخدام هذا الكتاب في أحد مقررات البرمجة ، قد يرغب الأستاذ في إضافة تمارين برمجة أخرى ، تعكس اهتمامات خاصة بأحد فروع العلم لمشاكل البرمجة .

والمعالم الأساسية للغة ملخصة في الملاحق من A إلى G ، وموجودة في نهاية الكتاب . ويجب أن يتكرر استخدام هذه المادة كدليل معد ، وللتذكير السريعة . وتكون مفيدة بصفة خاصة عند كتابة أو تصحيح برامج جديدة .

وأخيرا ... فإن القارئ الذي ينتهي من قراءة هذا الكتاب ؛ يصبح لديه فهم متماسك لمفاهيم البرمجة الأساسية ، والقواعد لغة البسكال الخاصة . ومن هذا يستطيع أن يكتسب فهما واقعياً لإمكانات ومحددات الكمبيوتر . كما يجب أن يكون قادرا أيضا على برمجة الكمبيوتر لأداء أنشطة محددة ، يقوم باختيارها بنفسه . كما يمكنه أن يمارس بعض الإثارات والإنعاش بنفسه ، بأن يصبح جزءا من ثورة الكمبيوتر الحالية ، والتي يبدو أنها ستكون عاملا أساسياً في إعادة تكوين المجتمعات الصناعية الحديثة .

بايرون س . جوتفريد

## المحتويات

صفحة	
١٥	<b>الفصل الأول : مفاهيم أولية</b>
١٥	١- مقدمة لأجهزة الكمبيوتر
١٦	٢- خواص الكمبيوتر
١٨	٣- طرق التشغيل
٢٣	٤- أنواع لغات البرمجة
٢٤	٥- مقدمة للغة البسكال
٢٧	٦- الخواص الجيدة للبرنامج
٢٩	<b>الفصل الثاني : أساسيات البسكال</b>
٢٩	١- مجموعة رموز البسكال
٣١	٢- الكلمات المحجوزة
٣٢	٣- المرفقات
٣٣	٤- المرفقات القياسية
٣٣	٥- الأعداد
٣٦	٦- السلاسل
٣٧	٧- أنواع البيانات
٣٨	٨- الثوابت
٣٩	٩- المتغيرات
٤٠	١٠- التعبيرات
٤١	١١- العبارات
٤٢	١٢- الإجراءات والدوال
٤٤	١٣- الرسومات التكوينية في لغة البسكال
٥١	<b>الفصل الثالث : بيانات من النوع البسيط</b>
٥١	١- بيانات من النوع الصحيح
٥٣	٢- بيانات من النوع الحقيقي
٥٤	٣- بيانات من النوع الحرفي
٥٧	٤- بيانات من نوع بوليان
٦٠	٥- الثوابت القياسية
٦٠	٦- الدوال القياسية
٦٢	٧- المزيد عن التعبيرات
٦٤	٨- عبارة التحديد
٧٣	<b>الفصل الرابع : إدخال وإخراج البيانات</b>
٧٣	١- ملفات المدخلات والمخرجات
٧٤	٢- عبارة اقرأ
٧٦	٣- عبارة READLN
٧٨	٤- دالتا EOF و EOLN
٧٨	٥- عبارة اكتب
٨٠	٦- عبارة WRITELN

٨٢	٧ - المخرجات المشكلة
٨٤	٨ - ملاحظات أخيرة
٩٥	<b>الفصل الخامس : إعداد وتشغيل برنامج بسكال كامل</b>
٩٥	١ - تخطيط برنامج البسكال
٩٧	٢ - كتابة برنامج بسكال
١٠١	٣ - إدخال البرنامج داخل الكمبيوتر
١٠٢	٤ - ترجمة وتنفيذ البرنامج
١٠٥	٥ - تشخيص الخطأ
١٠٨	٦ - تصحيح المنطق
١١٥	<b>الفصل السادس : مكونات التحكم</b>
١١٥	١ - مبادئ
١١٧	٢ - مكوّن « بينما .. اعمل »
١١٩	٣ - مكوّن « كرّر .. حتى »
١٢٢	٤ - مكوّن FOR
١٢٥	٥ - مكوّنات التحكم المتداخلة
١٢٨	٦ - مكوّن إذا
١٣٦	٧ - مكوّن الحالة
١٤٣	٨ - عبارة اذهب إلى
١٦١	<b>الفصل السابع : الإجراءات والدوال</b>
١٦١	١ - الإجراءات
١٦٤	٢ - مدى المعرفة
١٦٩	٣ - المؤشرات
١٧٩	٤ - الدوال
١٨٨	٥ - المزيد عن المؤشرات
١٩٢	٦ - إعادة الذاتية
٢١٩	<b>الفصل الثامن : البيانات البسيطة التي يعرفها المستخدم</b>
٢١٩	١ - بيانات من النوع المتعدد
٢٢١	٢ - بيانات من نوع المدى الجزئي
٢٢٢	٣ - المزيد عن التوضيحات
٢٢٣	٤ - استخدام البيانات التي يعرفها المستخدم
٢٣٥	<b>الفصل التاسع : المنظومات</b>
٢٣٥	١ - منظومات ذات بعد واحد
٢٤٣	٢ - منظومات متعددة الأبعاد
٢٤٩	٣ - عمليات تجرى على محتويات المنظومة
٢٥٤	٤ - منظومات مضغوطة
٢٥٦	٥ - السلاسل ومتغيرات السلاسل
٢٦٥	٦ - مؤشرات منظومة متغيرة الطول

٢٨٩	الفصل العاشر : السجلات
٢٨٩	١ - تعريف السجل
٢٩٤	٢ - تشغيل السجل
٢٩٨	٣ - مكوّن WITH
٣٠٧	٤ - سجلات متغيرة
٣٣٣	الفصل الحادي عشر : الملفات
٣٣٣	١ - أساسيات
٣٣٧	٢ - تعريف الملف
٣٣٧	٣ - إنتاج الملف
٣٤١	٤ - المزيد عن عبارة WRITE
٣٤٤	٥ - قراءة الملف
٣٤٦	٦ - المزيد عن عبارة READ
٣٤٧	٧ - حذف الملف
٣٦٠	٨ - ملفات النصوص
٣٨٥	الفصل الثاني عشر : الفئات
٣٨٥	١ - تعريف نوع الفئة
٣٨٧	٢ - عمل الفئات
٣٨٩	٣ - العمليات التي تجرى علي الفئات
٣٩١	٤ - مقارنة الفئات
٣٩٧	٥ - اختبار العضوية
٤١٣	الفصل الثالث عشر : القوائم والمشيريات
٤١٣	١ - أساسيات
٤١٦	٢ - تعريفات النوع
٤١٧	٣ - توضيحات المتغيريات
٤١٨	٤ - العمليات مع المتغيرات المشيرة ومتغيريات الإشارة
٤٢٢	٥ - إنتاج وإلغاء متغيرات ديناميكية
٤٤٩	ملحق (ا) : الكلمات المحجوزة
٤٤٩	ملحق (ب) : المعرفات القياسية
٤٥٠	ملحق (ج) : الإجراءات القياسية
٤٥١	ملحق (د) : الدوال القياسية
٤٥٢	ملحق (هـ) : المؤثرات
٤٥٣	ملحق (و) : الرسومات التكوينية
٤٦٠	ملحق (ز) : فئة رموز ASCII
٤٦١	إجابات المشاكل المتكاملة
٤٨٧	قائمة بأهم المصطلحات





## الفصل الأول

### مفاهيم أولية

## Introductory Concepts

يقدم هذا الكتاب تعليمات في البرمجة باستخدام لغة برمجة مرتبة ومنظمة ، تسمى بـ Pascal . وسوف نرى كيف يمكن تحليل وتخطيط ونقل مشكلة موصوفة في البداية بطريقة غير واضحة إلى برنامج بـ Pascal . وهذه المفاهيم موضحة بالتفصيل عن طريق العديد من عينات المشاكل الموجودة في الكتاب .

### ١ - مقدمة لأجهزة الكمبيوتر 1.INTRODUCTION TO COMPUTERS

أجهزة الكمبيوتر الحالية متعددة الأشكال والأحجام والتكلفة . فتستخدم أجهزة الكمبيوتر العملاقة متعددة الأغراض في الشركات الكبيرة والجامعات والمستشفيات والإدارات الحكومية ، وذلك لأداء حسابات علمية ، وحسابات أعمال معقدة . وعادة ما يشار إلى هذه الأجهزة على أنها أجهزة كمبيوتر كبيرة mainframes . وهي مرتفعة الثمن ( بعضها يقدر ثمنه بملايين الدولارات ) ، كما أنها تحتاج إلى علميين ومهندسين ومحاسبين معدين لاستخدامها .

وتستخدم أجهزة الكمبيوتر الكبيرة منذ بداية الخمسينيات ، إلا أن قلة من الأفراد أتاحت لهم الفرصة لاستخدامها ، وبصفة خاصة في السنوات الأولى من استخدامها . وعلى هذا ... فليس من المدهش أن ينظر عامة الناس نظرة غامضة - وبها بعض الارتباك - إلى الكمبيوتر .

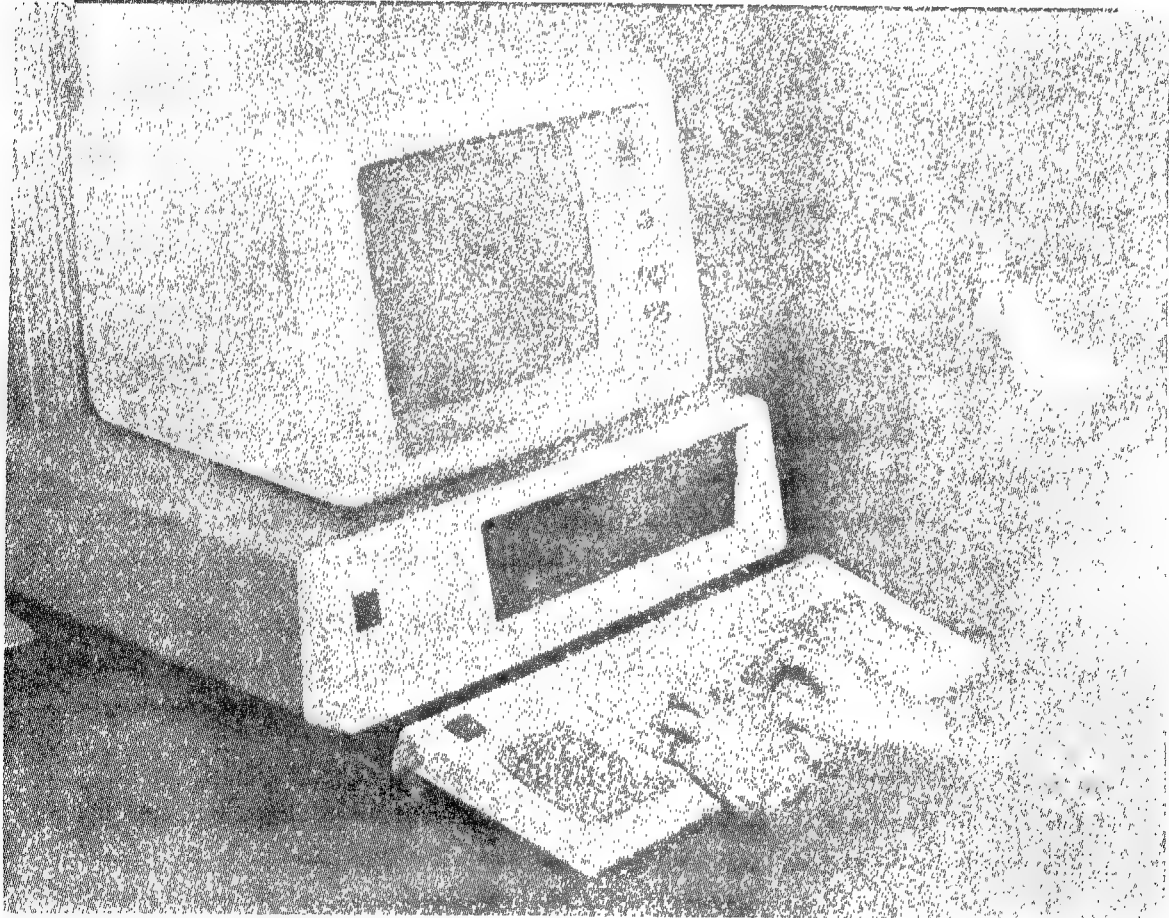
وقد رأت نهاية الستينيات وبداية السبعينيات تطورا لأجهزة ميني كمبيوتر minicomputers ، أصغر حجما وأقل تكلفة . وتقدم الكثير من هذه الآلات نفس أداء أجهزة الكمبيوتر الكبيرة ، وذلك بتكلفة أقل كثيرا من تكلفة هذه الأجهزة . وقد حصل الكثير من الأعمال والمؤسسات التعليمية ، التي ليس لديها المقدرة على الحصول على أجهزة الكمبيوتر الكبيرة ، على أجهزة ميني كمبيوتر ، مع استمرار توفر هذه الأجهزة .

وفي منتصف السبعينيات نتج عن تطور تقنية الدوائر المتكاملة ( رقائق السيليكون ) تطوير لأجهزة كمبيوتر أكثر صغرا ، وأقل تكلفة ، تسمى أجهزة ميكرو كمبيوتر microcomputer . وتتكون هذه الآلات من دوائر متكاملة ، وهي ليست أكبر ( أو أكثر تكلفة ) من آلة كاتبة تقليدية . ويمكن أن تستخدم في تطبيقات واسعة متعددة ، شخصية ، أو تعليمية ، أو تجارية ، أو تقنية . ويميل استخدامها لاستكمال استخدام أجهزة الكمبيوتر الكبيرة ، وليس لإحلال محلها . وفي واقع الأمر ... فإن العديد من المنظمات الكبيرة تستخدم أجهزة الميكرو كمبيوتر كنهايات طرفية أو محطات عمل متصلة بكمبيوتر كبير ( أو بسلسلة من أجهزة الكمبيوتر الكبيرة ) ، وذلك من خلال شبكة اتصالات (٥) .

وتطور الكمبيوتر الشخصي Personal Computer يحظى باهتمام خاص ، حيث إن هذه الأجهزة صغيرة ورخيصة ، وتميل إلى أن يستخدمها شخص واحد في نفس الوقت ( انظر الشكل ١ - ١ ) . وسعة العديد من هذه الآلات تقترب من سعة أجهزة الميني كمبيوتر . وأكثر من هذا ... يستمر الانخفاض في تكلفتها بصورة كبيرة مع تحسن كبير في أدائها . وتستخدم أجهزة الكمبيوتر الشخصية حاليا في الكثير من المدارس ونظم الأعمال الصغيرة . ويبدو أنها سوف تصبح أحد العناصر المنزلية الهامة .

---

(٥) لقد دخلت أسواق الكمبيوتر ، مع نهاية عام ١٩٨٩ ، أجهزة ميكرو كمبيوتر لها امكانيات تقترب من امكانيات أجهزة الكمبيوتر الكبيرة . ( المترجم ) .



الشكل رقم (١ - ١)

## 2 - COMPUTER CHARACTERISTICS

### ٢ - خواص الكمبيوتر

كل أجهزة الكمبيوتر عبارة عن وحدات إلكترونية ، يمكنها نقل وتخزين ومعالجة المعلومات ( أى البيانات ) . وهناك نوعان أساسيان مختلفان من البيانات ، وهى : البيانات العددية ، والبيانات الحرفية ( الأسماء ، والعناوين ، وما إلى ذلك ) . وتتطلب التطبيقات العلمية والتقنية تشغيل بيانات عددية أساسا ، بينما تشتمل تطبيقات الأعمال عادة على تشغيل كل من البيانات العددية والحرفية . وتستخدم بعض أجهزة الكمبيوتر فى تشغيل بيانات حرفية فقط ( مثل إعداد الخطابات والكتب وخلافه ) . وهذا ما يعرف بتشغيل الكلمات Word Processing

وتشغيل مجموعة بيانات محددة ، يجب أن يعطى الكمبيوتر مجموعة تعليمات مناسبة تسمى برنامجا Program . ويتم إدخال هذه التعليمات داخل الكمبيوتر لتخزن بعد ذلك فى ذاكرة الكمبيوتر

ويمكن تنفيذ البرنامج المخزن فى أى وقت . وهذا يتسبب فى حدوث الأشياء التالية :

١ - يتم إدخال مجموعة من المعلومات ، تسمى بيانات مدخلات Input Data داخل الكمبيوتر ( عن طريق نهاية طرفية ، أو قارئ بطاقات ، أو غيرها ) . وتخزن فى جزء من أجزاء ذاكرة الكمبيوتر

٢ - بعد ذلك يتم تشغيل بيانات المدخلات لإنتاج نتائج معينة ، تعرف بأنها بيانات المخرجات Output Data

٣ - تطابع بيانات المخرجات ( وربما يطابع معها بعض بيانات المدخلات ) على أوراق ، أو تظهر على شاشة بحسب monitor ( أى وحدة عرض مرئي ) .

ويمكن إعادة هذه الخطوات الثلاثة العديد من المرات ، إذا كانت هناك رغبة فى ذلك لتشغيل كمية بيانات كبيرة فى تسلسل سريع . ويجب - على أية حال - فهم أن كل من هذه الخطوات يمكن أن تكون طويلة ومعقدة ، وبصفة خاصة الخطوات رقم ٢ ، ورقم ٣ .

مثال (١-١)

تم برمجة كمبيوتر لحساب مساحة الدائرة باستخدام العلاقة :  $A = \pi r^2$  ، وذلك إذا ما أعطيت القيمة العددية لنصف القطر كبيانات مدخلات . يتطلب ذلك اتباع الخطوات التالية :

١ - قراءة قيمة عددية لنصف قطر الدائرة .

٢ - حساب قيمة المساحة ، باستخدام العلاقة سابقة الذكر ( سوف تخزن هذه القيمة مع بيانات المدخلات فى ذاكرة الكمبيوتر ) .

٣ - طباعة ( أو عرض ) قيمة نصف القطر ، وقيمة المساحة المناظرة له .

٤ - التوقف .

تتطلب كل خطوة من هذه الخطوات إحدى تعليمات - أو العديد من تعليمات - برنامج الكمبيوتر .

نوضح المناقشة السابقة سمتين هامتين للكمبيوتر ، هما : الذاكرة ، وإمكانية البرمجة . كما أن السرعة والاعتمادية من السمات الهامة الأخرى . وسوف نذكر المزيد عن الذاكرة والسرعة والاعتمادية فى المقاطع القليلة التالية . أما موضوع إمكانية البرمجة ، فسوف يناقش بالتفصيل خلال مانبقى من هذا الكتاب .

الذاكرة : Memory

كل المعلومات المخزنة داخل ذاكرة الكمبيوتر ، تخزن على هيئة شفرة من خليط من الأصفار والرقم 1 . وتسمى هذه الشفرة ( أصفار وأرقام 1 ) بالبت ، وهى اختصار للأرقام الثنائية binary digits ، وتمثل كل بت تمثيلاً إلكترونياً بحيث إنها تكون فى الوضع الموصل On الذى يمثل الرقم 1 . أو وضع الفصل Off الذى يمثل الصفر .

ومعظم أجهزة الكمبيوتر الصغيرة لها ذاكرات مرتبة طبقاً لتكرار كل ثمانية بت . وتسمى كل ثمانية بت بالبايت byte . وعادة ما يمثل البايت الواحد رمزا واحداً ( أى يمثل حرفاً ، أو رقماً ، أو رمزا خاصاً ) . ويمكن أن تحتل إحدى التعليمات 1 أو 2 أو 3 بايت ، كما أن الكميات العددية يمكنها أن تحتل من 2 إلى 8 بايت ، وذلك طبقاً لدقة العدد ونوعه .

وعادة ما يعبر عن حجم سعة ذاكرة الكمبيوتر بمضاعفة  $2^{10}$  ، وهو 1024 بايت . ويشار إلى هذا العدد بالرمز 1 KB . وعادة ما تتراوح سعة ذاكرة أجهزة الكمبيوتر الصغيرة من 64 KB إلى 1024 KB ، أى إلى 1 MB (\*) .

مثال (٢-١)

سعة جهاز كمبيوتر شخصى صغير هى 64 KB . وعلى هذا ... فيمكن تخزين  $65,536 = 64 \times 1024$  رمزا أو عدداً من التعليمات فى ذاكرة هذا الكمبيوتر . فإذا ما استخدمت الذاكرة لتمثيل بيانات حرفية ، فيمكن تخزين حوالى 800 اسم وعنوان داخل ذاكرة الكمبيوتر فى آن واحد ( وذلك بفرض أن كل اسم وعنوان يشغلان 80 خانة ) .

(\*) ازدادت هذه السعة مع نهاية عام ١٩٨٩ ليكن أن تصل إلى أكثر من 200 MB ( المترجم ) .

أما أجهزة الكمبيوتر الكبيرة ، فلهذا ذاكرات دخلت على هيئة كلمات Words ، وليس على هيئة بايت ، يتمتع كل كلمة على عدد كبير نسبيا من البت ، وعادة ما يكون 32 أو 36 . وهذا يسمح بتمثيل مجموعة بسيطة من الحروف ( عادة أربعة أو خمسة ) داخل ذاكرة مكونة من كلمة واحدة . وعادة ما يميز عن سعة أجهزة الكمبيوتر الكبيرة بدفعات (  $10^3 = 1024$  k words ) ، فجهاز الكمبيوتر الكبير يمكن أن تكون سعة ذاكرته العديد من ملايين من الكلمات .

### مثال (١-٢)

سعة ذاكرة جهاز كمبيوتر ذي أغراض عامة ، هي 2048 KW ، وهي تتألف  $2048 \times 1024 = 2,097,152$  كلمة . فإذا ما استخدمت الذاكرة لتمثيل بيانات عديدة ، فيمكن تخزين حوالي ٢ مليون عدد داخل الكمبيوتر في أي وقت . أما إذا استخدمت الذاكرة لتمثيل بيانات حرفية ، بدلا من البيانات العددية ، فيمكن تخزين حوالي ٨ مليون حرف في نفس الوقت . وهذه الذاكرة تكفي لتخزين مستويات كتاب كامل ، بل تزيد عن ذلك أيضا .

كما تستخدم معظم أجهزة الكمبيوتر وحدات تخزين مساعدة auxiliary Storage Devices أيضا ( مثل شرائط المغناطيسية ، والأقراص المغناطيسية ، وغيرها ) ، وذلك بالإضافة إلى ذاكرتها الابتدائية . وعادة ماتراوح سعة هذه الوحدات من عدة مئات من الآلاف من البايت ( بالنسبة لأجهزة الكمبيوتر الصغيرة ) إلى عدة ملايين من الكلمات ( بالنسبة لأجهزة الكمبيوتر الكبيرة ) . وأكثر من هذا ... فإنها تسمح بتسجيل المعلومات تسجيلا دائما ، حيث يمكن تركيبها على الكمبيوتر ، أو فكها منه وتخزينها لحين الحاجة إليها . وعلى أية حال ... فإن زمن الاتصال ( وهو الزمن اللازم لتخزين المعلومات أو لاسترجاعها ) يكون كبيرا لهذه الوحدات الثانوية ، بالمقارنة لقربه الخاص بالذاكرة الابتدائية .

### السرعة والاعتمادية : Speed and Reliability

نظرا لسرعة الكمبيوتر المرتفعة جدا ، فإنه يستطيع أداء الحسابات التي تحتاج إلى أشهر لادائها يدويا في عدة دقائق . والأنشطة البسيطة ، مثل جمع عددين ، يمكن أن تؤدي في جزء من الميكروثانية (  $10^{-6}$  ميكروثانية = ثانية ) . وعلى معظم المستويات العامة ، فإن درجات الدلالة في نهاية الفصل الدراسي في إحدى الجامعات الكبيرة ، يمكن إعدادها في عدة دقائق بالنسبة للكمبيوتر .

وإن السرعة العالية جدا يجعلها درجة اعتمادية على نفس المستوى . فالكمبيوتر لا يخطئ عمليا على الإطلاق . وأخطاء الكمبيوتر التي يركز عليها هي وسائل الإعلام ، مثل تسليم أحد الأفراد فاتورة تفيد بأن مديونيته وصلت مليون دولار أحد محلات البيع بالتجزئة ، وهي إلا نتيجة لخطأ في البرمجة أو في النقل ، وليست بسبب خطأ للكمبيوتر نفسه .

## 3 MODES OF OPERATION

### ٣ - طرق التشغيل

هناك طريقتان مختلفتان يمكن أن تستخدم بهما وسائل الكمبيوتر . هاتان الطريقتان هما ، طريقة الدفعة وطريقة التداخل . وكل من الطريقتين شائعة الاستخدام . كما أن كل منهما لها سميراتها بالنسبة لنوع معين من المشاكل .

### Batch Processing

### تشغيل الدفعة

في الأيام الأولى لاستخدام الكمبيوتر ، كانت كل الأعمال تنفذ عن طريق تشغيل الدفعة batch processing ومازالت هذه الطريقة مستخدمة حتى الآن ، بالرغم من قلة استخدامها بالنسبة لما كانت عليه في الأيام الأولى لاستخدام الكمبيوتر .

وفي تشغيل الدفعة ، يقرأ العديد من الأعمال داخل الكمبيوتر ، ويتخزن هذه الأعمال داخلياً ، ثم يتم تشغيلها على التوالي . ( وتشير الأعمال إلى برامج كمبيوتر ، وبإحصائيتها من مجموعات من بيانات المدخلات التي يجري عليها التشغيل ) . ويتطلب تشغيل الدفعة أن تكون البرامج والبيانات مسجلة على بطاقات مثبته . وتقرأ هذه المعلومات داخل الكمبيوتر بواسطة قارئ البطاقات ، ثم بعد ذلك يتم تشغيلها . وبعد الانتهاء من تشغيل أحد الأعمال ؛ تطبع المخرجات معها قائمة بالبرنامج على شريط طويل من الأوراق بواسطة طابع ذي سرعة طباعة عالية .

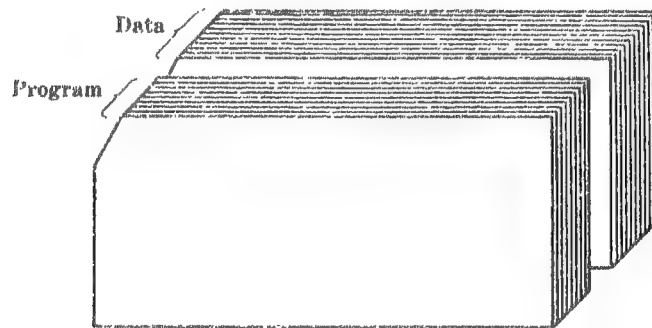
أما تشغيل الدفعة الحديث ، فيرتبط بصفة عامة بنظام المشاركة الزمنية . وفي هذا النظام يتم إدخال البرنامج ومجموعة البيانات داخل الكمبيوتر عن طريق نهاية طرفية أو ميكروكمبيوتر . وبعد ذلك تخزن المعلومات داخل ذاكرة الكمبيوتر ، وتنفذ طبقاً لترتيبها المناسب . وتفضل هذه المصيفة من صيغ تشغيل الدفعة عن الطريقة التقليدية ، حيث إنها تلغي الحاجة إلى البطاقات المثقبة ، وتسمح بتنقيح معلومات المدخلات ( البرامج والبيانات ) أثناء إدخالها .

ويمكن نقل كميات كبيرة من المعلومات ( البرامج والبيانات ) داخل الكمبيوتر ، ومنه إلى الخارج بسرعة عالية في تشغيل الدفعة . والأكثر من هذا هو أن المستفيد لا يكون في حاجة لأن يتواجد أثناء تنفيذ العمل الخاص به . وعلى هذا .. فإن مثل هذه الطريقة تناسب الأعمال التي تتطلب وقتاً طويلاً من وقت الكمبيوتر ، أو أنها تكون طويلة وناحية أخرى ... قد يختلف إجمالي الوقت اللازم لتشغيل أحد الأعمال بهذه الطريقة من عدة دقائق إلى العديد من الساعات ، وذلك بالرغم من أن هذا العمل قد لا يحتاج إلا إلى ثانية واحدة أو اثنتين من الوقت الفعلي لتشغيل الكمبيوتر ( يجب أن ينتظر كل عمل دوره قبل أن يمكن قراءته وتشغيله وطباعة مخرجاته ) . وعلى هذا ... فقد لا يكون مرغوباً فيه تشغيل الدفعة عندما يكون هناك حاجة إلى تشغيل العديد من الأعمال الصغيرة البسيطة ، والحصول على النتائج بأسرع ما يكون .

#### مثال (١-٤)

لدى أحد الطلاب ١٠٠٠ قيمة مختلفة لنصف قطر دائرة ، ويريد أن يحسب مساحة الدائرة لكل قيمة من هذه القيم . لعمل ذلك ... يجب عليه أن ينفذ برنامج كمبيوتر يشبه البرنامج الذي سبق وصفه في المثال (١ - ١) . وتجرى الحسابات في هذه الحالة ١٠٠٠ مرة ، بمعدل مرة واحدة لكل قيمة من قيم نصف القطر ( ويجب أن يعدل البرنامج ليتمكن تكرار تنفيذه ) . يستخدم تشغيل الدفعة ، حيث إن كل الحسابات سوف تجرى مرة واحدة . دعنا نفترض أن البرنامج والبيانات على بطاقات مثبته .

لتشغيل البيانات ، سوف يقرأ الطالب مجموعة البطاقات داخل الكمبيوتر . وأول جزء من مجموعة البطاقات يحتوي على البرنامج . يلي البرنامج ١٠٠٠ بطاقة بيانات ، كل منها يحتوي على قيمة واحدة لنصف القطر . ومجموعة البطاقات هذه موضحة في الشكل رقم (١-٢) .



الشكل رقم (١-٢)

وبعد قراءة مجموعة العلاقات داخل الكمبيوتر ، قد يحدث تغيير ، خاصة سمات قبل أن ينفذ البرنامج . ويتم تشغيل البيانات . وبعد الانتهاء من إجراء الحسابات ، تطابع النتائج على شريط خبير «ن الأوراق» وفي هذه الحالة يطلق الطالب قائمة تحتوي على البرنامج الفعلي ، ويتبعه ١٠٠٠ نجا من الأعداد . كل زوج من هذه الأزواج من الأعداد يمثل قيمة واحدة لذو نصف القطر ، ومساحة الدائرة المتناثرة لها . يبين الشكل رقم (١-٣) جزءاً من مخرجات الكمبيوتر .

RADIUS	AREA
1	3.141593
2	12.566372
3	28.274337
4	50.265488
5	78.5398251
6	113.097348
7	153.938057
8	201.061952
9	254.469033
10	314.1593
11	380.132753
12	452.389392
13	530.929217
14	615.752229
15	706.858425
16	804.247808
17	907.920378
18	1017.87613
19	1134.11507
20	1256.6372

الشكل رقم (١-٣)

## Interactive Computing

## التشغيل التداخلي

يحدث التشغيل التداخلي Interactive Computing باستخدام كمبيوتر شخصي ، مثل الموضح في الشكل رقم (١ - ١) ، أو باستخدام نهاية هارفية الكمبيوتر كبير ، كما في الشكل رقم (١ - ٢) . وفي أي من حالتين ، يقدم المستخدم معلومات المدخلات للكمبيوتر من خلال لوحة المفاتيح التي تشبه مقاييس الآلة الكاتبة . ومعلومات المخرجات المتناثرة يتم طباعتها على شريط الحويل من الأوراق ، أو تعرض على موجه برقي ( قد يكون من الأنواع التي تعمل على «مخرجات مطبوعة لبعض التطبيقات» ، حيث إن ذلك يقدم نسخة دائمة من التشغيل التداخلي . وعلى أية حال . . فإن استخدام الموجه عادة ما يكون أكثر راحة ) . وعادة ما يشار إلى النهايات الطرفية المستخدمة في التشغيل التداخلي ، بأنها نهايات طرفية مرتبطة Console .

إحدى السمات المعنوية للتشغيل التداخلي هي أن المستخدم يكون قادراً على إجراء «حوار» مع الكمبيوتر أثناء التشغيل . وعلى هذا . . . فسيكون للمستخدم أن يسأل الكمبيوتر بصفة دورية أن يقدم له معلومات معينة تحدد الإجراء التالي الذي سيتخذه الكمبيوتر .





الشكل رقم (١-٤)

### مثال (١-٥)

يرغب أحد الطلاب في استخدام الكمبيوتر في حساب نصف قطر الدائرة التي مساحتها ١٠٠ . والبرنامج الذي يحسب مساحة الدائرة إذا ما أعطى نصف القطر موجود . ( لاحظ أن الموجود هو عكس ما يريده الطالب تماماً ) . وهذا البرنامج ليس هو البرنامج المطلوب ، إلا أنه يسمح للطالب أن يعمل عن طريق التجربة والخطأ ، حتى يجد قيمة نصف قطر الدائرة التي تقترب مساحتها من ١٠٠ .

وبمجرد إدخال البرنامج المطلوب تظهر الرسالة التالية .

RADIUS=?

عند ذلك يقوم الطالب بإدخال قيمة نصف القطر . دعنا نفترض أن الطالب قام بإدخال قيمة ٥ لنصف القطر . عند ذلك يستجيب الكمبيوتر للطباعة .

AREA= 78.5398

DO YOU WISH TO REPEAT THE CALCULATION?

عند ذلك يكتب الطالب "yes" أو "no" ، فإذا ماكتب الطالب "yes"؛ تظهر له الرسالة التالية :

RADIUS=?

وتعاد نفس الإجراءات . أما إذا ماكتب الطالب "no" ؛ فتظهر له الرسالة التالية :

GOODBYE

ويفصل البرنامج عن الكمبيوتر .

ويوضح الشكل رقم (٥-١) المعلومات التي تطبع أثناء عملية متداخلة تقليدية باستخدام البرنامج سالف الذكر ، مع وضع خط تحت الكلمات التي كتبها الطالب . وقد تحددت قيمة تقريبية لنصف القطر تساوى ٥,٦ بعد ثلاث محاولات .

#### INTRODUCTORY CONCEPTS

RADIUS=? 5

AREA= 78.5398

DO YOU WISH TO REPEAT THE CALCULATION? YES

RADIUS=? 6

AREA= 113.097

DO YOU WISH TO REPEAT THE CALCULATION? YES

RADIUS=? 5.6

AREA= 98.5204

DO YOU WISH TO REPEAT THE CALCULATION? NO

GOODBYE

#### الشكل رقم (٥-١)

لاحظ الطريقة التي تبدو كحوار بين الطالب والكمبيوتر . ولاحظ أيضاً أن الطالب ينتظر حتى يرى قيمة المساحة المحسوبة ، قبل أن يقرر ما إذا كانت ستجرى حسابات أخرى أم لا . فإذا ما بدأت عملية حسابات أخرى ، فتعتمد القيمة الجديدة لنصف القطر الذي يقدمها الطالب على النتائج التي سبق الحصول عليها .

وفى بعض الأحيان ، يقال عن البرامج التي تصمم لتطبيقات التشغيل المتداخل بأنها ذات طبيعة حوار Conversational . والمباريات التي تجرى مع الكمبيوتر - مثل لعب الشطرنج - هي أمثلة ممتازة لمثل تطبيقات التداخل هذه .

## Timesharing

## المشاركة الزمنية

المشاركة الزمنية timesharing هي صيغة من صيغ التشغيل المتداخل ، والتي يكون العديد من المستخدمين قادرين فيها على استخدام كمبيوتر واحد في نفس الوقت . ويتصل كل مستفيد بالكمبيوتر من خلال نهاية طرفية ، مثل النهاية الموضحة في الشكل رقم (١-٤) ، أو من خلال ميكرو كمبيوتر ، مثل الموضح في الشكل رقم (١-١) . ويمكن أن تكون النهايات الطرفية متصلة سلكيا بالكمبيوتر مباشرة ، أو يمكن أن تكون متصلة بالكمبيوتر عبر خطوط الهاتف ، أو عبر نواثر ميكرويف . وعلى هذا ... فيمكن للنهاية الطرفية أن توضع بعيدا عن الكمبيوتر المستخدم . وربما تصل هذه المسافة إلى العديد من مئات الأميال .

وحيث إن الكمبيوتر يعمل بطريقة أسرع من عمل النهاية الطرفية ، فيمكن لجهاز كمبيوتر واحد أن يخدم عدداً كبيراً من النهايات الطرفية في نفس الوقت . وعلى هذا ... لا يكون أى مستفيد قلقاً من ناحية تواجد مستفيدين آخرين . وسوف يبدو له أن الكمبيوتر كما لو كان يعمل له هو بمفرده .

وتناسب المشاركة الزمنية تشغيل الأعمال الصغيرة نسبياً ، والتي لا تتطلب نقلاً كبيراً للبيانات ، أو تتطلب وقتاً كبيراً من الكمبيوتر . ومعظم تطبيقات الكمبيوتر التي تظهر في المدارس والجامعات والمكاتب التجارية تتميز بمثل هذه السمات . ويمكن تشغيل مثل هذه التطبيقات بسرعة وبسهولة وبتكلفة أقل إذا ما استخدم تشغيل المشاركة الزمنية .

### مثال (١-٦)

لدى إحدى الجامعات الكبيرة إمكانية المشاركة الزمنية في كمبيوتر محتوٍ على ١٠٠ نهاية طرفية للمشاركة الزمنية . وتوجد هذه النهايات الطرفية في المواقع المختلفة للجامعة . وتتصل النهايات الطرفية بجهاز كمبيوتر عبر خطوط الهاتف . وتنقل كل نهاية طرفية بيانات إلى الكمبيوتر أو منه ، بسرعة قصوى تعادل ١٢٠ رمزا في الثانية . ويمكن استخدام جميع النهايات الطرفية في نفس الوقت ، بالرغم من أنها متداخلة مع كمبيوتر واحد .

بالإضافة إلى ذلك ... تتصل بالكمبيوتر ٢٠ نهاية طرفية ، تقع في أماكن بعيدة ، حيث يوجد خمسة عشر من هذه النهايات الطرفية في خمسة مواقع في منطقة تواجد الكمبيوتر . أما الخمس نهايات الطرفية الأخرى ، فتقع في معامل أبحاث حكومية . ويمكن استخدام كل النهايات الطرفية - والبالغ عددها ١٢٠ - في نفس الوقت ( وعادة ما يحدث ذلك ) . وبالمشاركة بهذه الطريقة في الكمبيوتر ، يكون كل مستفيد قادراً على استخدام خدمة الكمبيوتر الكبير بتكلفة معقولة .

ويمكن دمج بعض معالم تشغيل الدفعة والمشاركة الزمنية إذا ما كانت هناك رغبة في ذلك . فيمكن على سبيل المثال ... إدخال مجموعة من بيانات المدخلات من نهاية طرفية مباشرة ، على أن يتم تشغيلها بطريقة الدفعة الذي سبق وصفها . وهناك إمكانية أخرى ، وهي استخدام قارئ بطاقات ( تشغيل الدفعة ) في إدخال برنامج ومجموعة بيانات ، ثم تعديل البرنامج ، وتشغيل البيانات بطريقة المشاركة الزمنية . مثل هذه العمليات أصبحت شائعة الاستخدام في الوقت الحديث .

## ٤ - أنواع لغات البرمجة 4- TYPES OF PROGRAMMING LANGUAGES

هناك العديد من اللغات المختلفة التي يمكن استخدامها في برمجة الكمبيوتر . وأكثر هذه اللغات قرباً للكمبيوتر هي لغة الآلة machine language . وهي مجموعة من التعليمات التفصيلية جداً الخفية ، التي تتحكم في نواثر الكمبيوتر الداخلية . وهذه هي اللهجة الطبيعية للكمبيوتر . ويكتب القليل جداً من برامج الكمبيوتر في واقع الأمر بهذه اللغة لسبيين : أولاً : أن لغة الآلة مرهقة جداً في العمل بها . وثانياً : أن معظم أجهزة الكمبيوتر لديها تعليماتها الخاصة بها . وعلى هذا فإن البرنامج المكتوب بلغة الآلة لأحد أنواع أجهزة الكمبيوتر لا يمكن تشغيله على نوع آخر من أنواع أجهزة الكمبيوتر ، دون إدخال تعديلات جوهرية عليه .

وعادة مايكتب برنامج الكمبيوتر بإحدى اللغات مرتفعة المستوى high - level language ، والتي تكون تعليماتها أكثر توافقية مع اللغات الأدمية . ومعظم هذه اللغات مرتفعة المستوى هي لغات ذات أغراض عامة general - purpose ، مثل لغة البيسكال ، ( وهناك لغات ذات أغراض عامة أخرى ، مثل : البيسك ، والكويل ، والفورتران ، ولغة PL/1 ) . كما يوجد لغات أخرى مرتفعة المستوى ذات أغراض خاصة special - purpose ، والتي تصمم تعليماتها لتناسب أحد أنواع التطبيقات المحددة بصفة خاصة .

وكقاعدة عامة ... فإن إحدى تعليمات أى لغة ذات مستوى مرتفع ، تعادل عدة تعليمات بلغة الآلة . والأكثر من هذا ... فيمكن تشغيل البرنامج المكتوب بإحدى اللغات مرتفعة المستوى على العديد من أجهزة الكمبيوتر ، وذلك مع إدخال بعض التعديلات البسيطة ، أو بدون إدخال أى تعديلات بالمرّة . وعلى هذا ... فإن استخدام اللغات مرتفعة المستوى يقدم بعض المميزات المعنوية جدا لنا عن استخدام لغة الآلة ، وبالتحديد فإنه يقدم لنا البساطة والانتظام ، وإمكانية استخدام البرنامج مع العديد من أجهزة الكمبيوتر ( أى عدم اعتماد البرنامج على الكمبيوتر نفسه ) .

ويجب - على أية حال - أن يترجم البرنامج المكتوب بلغة مرتفعة المستوى إلى لغة الآلة قبل أن يكون قابلا للتنفيذ . وهذا مايعرف بالترجمة Compilation ، أو بالتفسير Interpretation طبقا لكيفية أداء عملية الترجمة . ( معظم صيغ البيسكال يتم لها عملية ترجمة ، وليست عملية تفسير ) .

وبصفة عامة ... فإنه من المريح أكثر إعداد البرنامج الجديد باستخدام مفسر ، بدلا من استخدام مترجم ، وذلك بالرغم من أن البرنامج المترجم ينفذ بصورة أسرع كثيرا من البرنامج المفسر . ( أسباب ذلك لاتقع فى مدى مناقشتنا الحالية ) ، وفى أى حالة من الحالتين - على أية حال - تتم عملية الترجمة تلقائيا داخل الكمبيوتر . وفى واقع الأمر لا يكون المبرمج المبتدئ مهتما بمثل هذه العملية على الإطلاق ، حيث إنه يرى البرنامج الأصلي فقط ، ويرى بيانات المدخلات ، وبيانات المخرجات التى يحصل عليها من البرنامج فقط .

والمترجم أو المفسر فى حد ذاته ماهو إلا برنامج كمبيوتر يقبل برنامج كمبيوتر مكتوبا بلغة مرتفعة المستوى كبيانات مدخلات ، وينتج عنه نفس البرنامج بلغة الآلة كمخرجات . وطبقا لذلك ... يسمى البرنامج الأصلي مرتفع المستوى ببرنامج المصدر Source ، ويسمى البرنامج الناتج بلغة الآلة ببرنامج التشغيل Object . ويجب أن يكون لكل جهاز كمبيوتر مترجمه أو مفسره الخاص بالنسبة للغة مرتفعة المستوى المستخدمة معه . وهذا الاستخدام للمترجمات أو المفسرات هو الذى يسمح لنا بتحقيق الانتظام وعدم الاعتماد على الكمبيوتر بالنسبة للغات مرتفعة المستوى ، مثل البيسكال .

## 5 - INTRODUCTION TO PASCAL

## ٥ - مقدمة للغة البيسكال

لغة البيسكال هي لغة تستخدم فى الأغراض العامة ، كما أنها لغة مرتفعة المستوى . وقد استخلصت هذه اللغة من لغة Algol-60 . وتعد تعليمات لغة البيسكال بطريقة تشبه التعبيرات الجبرية ، مستخدمة بعض كلمات اللغة الإنجليزية ، مثل الكلمات التالية :

BEGIN, END, read, write, IF, THEN, REPEAT, WHILE, DO.

وبهذا الشكل فإن لغة البيسكال تشبه العديد من اللغات ذات المستوى المرتفع . كما تحتوى لغة البيسكال على بعض المعالم الخاصة بها ، والتي سمعت خصيصا لتشجيع استخدام البرمجة المرتبة Structured Programming . والبرمجة المرتبة هي منهج مرتب ومنظم لإعداد برامج واضحة ، ومرتفعة الكفاءة ، وخالية من الأخطاء . ولهذا السبب يفضل العديد من المعلمين والمبرمجين المهنيين استخدام لغة البيسكال عن لغات الأغراض العامة الأخرى .

وقد سميت لغة البيسكال باسم العالم الفرنسى المشهور ( بليز بسكال ) الذى ولد فى عام ١٦٢٣م ، وتوفى فى عام ١٦٦٢م ، والذى حقق العديد من الاكتشافات العلمية ، والتي منها اختراع أول آلة حاسبة ميكانيكية فى العالم .

## History of Pascal

## نشأة لغة البسكال وتطورها

أعد لغة البسكال في بدايتها ( نيكولوس ويرث Niklaus Wirth ) في جامعة التقنية في زيورخ في بداية السبعينيات . وقد كان الهدف الأساسي لويرث هو تطوير لغة منظمة مرتفعة المستوى لتعليم البرمجة المرتبة . وأحيانا يشار إلى التعريف الأساسي للغة الذي أعده ويرث بأنه البسكال القياسية Standard Pascal ، أو البسكال القياسية كما يعرفها ويرث وجنسين<sup>(٢)</sup> Standard Pascal as defined by Jensen and Wirth . وعلى أية حال ... فهناك بعض الغموض في اصطلاح بسكال القياسية ، حيث إنه يوجد العديد من القياسات المختلفة في وقتنا الحالي .

وتختلف معظم مترجمات البسكال بعض الشيء عن التعريف الأساسي لويرث . فقد اقترحت المؤسسة العالمية للقياسات International Standards Organization (ISO/DIS7185) صيغة قياسية أوروبية حديثة . وفي فترة كتابة هذا الكتاب ، كانت هناك صيغة قياسية أمريكية تحت الإعداد بواسطة المعهد القومي الأمريكي للقياسات American National Standards Institute (ANSI) بالمشاركة مع معهد الهندسة الكهربائية والإلكترونية Institute of Electrical and Electronic Engineers (IEEE) . ويبدو أن الصيغة القياسية الأمريكية سوف تختلف اختلافا بسيطا عن الصيغة القياسية الأوروبية .

وحاليا تستخدم لغة البسكال بصورة واسعة في الولايات المتحدة الأمريكية وفي أوروبا ، وذلك كلفة تعليمية ولغة قوية للاستخدام في الأغراض العامة في العديد من التطبيقات المختلفة . وقد ازدادت شعبية استخدامها مع كل من أجهزة الكمبيوتر الكبيرة والصغيرة . وفي واقع الأمر ... فإن لغة البسكال أصبحت شائعة الاستخدام من قبل المستفيدين من أجهزة الميكروكمبيوتر وهناك بعض التوقعات بأنها سوف تحل محل لغة البيسك كلفة سائدة مع أجهزة الميكروكمبيوتر خلال السنوات القليلة التالية .

وهذا الكتاب موجه بصفة مبدئية نحو استخدام صيغتي البسكال القياسيتين ISO/ANSI . وحيث إن هاتين الصيغتين القياسيتين ليستا شائعتا الاستخدام ، فسوف نناقش السمات غير القياسية أيضا . وتقدم هذه المادة الأساس لكل المترجمات التجارية للغة البسكال . وعلى هذا ... فإن القارئ الذي يلم بهذه المادة لن يكون أمامه سوى صعوبة بسيطة ليتعلم أى صيغة أخرى من صيغ هذه اللغة .

## Structure of a Pascal Program

## ترتيب برنامج البسكال

يحتوي كل برنامج مكتوب بلغة البسكال على عنوان ومجموعة . ويبدأ العنوان بكلمة PROGRAM ، ويتبعها بعض المعلومات الإضافية المطلوبة . ويكتب هذا الجزء من البرنامج في سطر واحد فقط .

أما المجموعة block ، فتحتمل على جزئين أساسيين ، وهما : جزء التوضيح ، وجزء العبارات . ويعرف جزء التوضيح Declaration part عناصر البيانات المختلفة المستخدمة في البرنامج . أما جزء العبارات Statement part فيحتوي على العبارات الفعلية التي تتسبب في اتخاذ إجراءات . ويجب أن تظهر جملة واحدة على الأقل في كل برنامج مكتوب بلغة البسكال . والترتيب الشامل للبرنامج موضح بتفصيل أكثر في التخطيط التالي :

1. Header
2. Block
  - (a) Declarations
    - Labels
    - Constants
    - Type definitions
    - Variables
    - Procedures and functions
  - (b) Statements

(٢) Jensen, K. and N.Wirth "Pascal User Manual and Report", 2nd. edition, Springer - Verlag, 1974.

(٤) مامن احد ينكر الانتشار السريع للغة البسكال . الا انه من المتوقع ان يزداد الاقبال على كل من لغة البسكال ولغة البيسك خاصة بعد ادخال الكثير من التعديلات على صيغتها مثل بييسك السريع وتربويسك ( المترجم ) .

وسوف نناقش كل عنصر من هذه العناصر للبرنامج بتفاصيل أكثر فيما بعد في هذا الكتاب . وكل ما يهمنا الآن هو الشكل العام فقط . ويجب الإشارة إلى أنه ليست هناك حاجة لظهور مكونات جزء التوضيح كلها في كل برنامج من برامج البسكال . فإذا وجدت على أية حال ، فيجب أن تظهر بنفس الترتيب الموجود أعلاه .

### مثال (٧-١)

مساحة الدائرة . يوجد هنا برنامج أولى بلغة البسكال يسمى Circle ، ويقرأ نصف قطر الدائرة ، ويحسب مساحتها ، ثم يكتب قيمة كل من نصف القطر والمساحة .

```
PROGRAM circle(input,output);      (* HEADER *)
VAR area,radius : real;            (* VARIABLE DECLARATION *)
BEGIN
  read(radius);                    (* STATEMENT *)
  area := 3.14159*sqr(radius);      (* STATEMENT *)
  write(radius,area)               (* STATEMENT *)
END.
```

المكونات الأساسية لهذا البرنامج تم فصلها وتسميتها بالترتيب لتعظيم التنظيم الشامل للبرنامج . وفي العادة لا يأخذ البرنامج هذا الشكل . وبدلاً من ذلك ، فقد يظهر البرنامج على النحو التالي :

```
PROGRAM circle(input,output);
(* PROGRAM TO CALCULATE THE AREA OF A CIRCLE *)
VAR area,radius : real;
BEGIN
  read(radius);
  area := 3.14159*sqr(radius);
  write(radius,area)
END.
```

وتجب الإشارة إلى السمات التالية بالنسبة لهذا البرنامج :

- ١ - بعض الكلمات مكتوبة بحروف كبيرة . وهذه هي الكلمات المحجوزة reserved words أو الكلمات الأساسية Keywords ، والتي لها معنى سبق تحديده ( وسوف تذكر بالتفصيل فيما بعد ) .
- ٢ - يحتوى السطر الأول على اسم البرنامج (circle) ، وعلى بعض المعلومات الإضافية التي سوف يتم وصفها في القسم التالي . وهذا هو عنوان البرنامج .
- ٣ - السطر الثاني هو تعليق Comment يعرف الغرض من البرنامج . ويمكن تمييز التعليقات بصفة دائمة ، حيث إنها توضع بين رمزين خاصين هما : ( \* ... \* ) .
- ٤ - لاحظ الثلاثة أسطر المرحلة ، والموجودة بين BEGIN,END . هذه هي عبارات Statements البرنامج . فهي تتسبب في إدخال قيمة نصف القطر في الكمبيوتر ، وفي حساب قيمة المساحة ، وفي كتابة قيمة نصف القطر وقيمة المساحة . وترحيل هذه العبارات غير ضروري ، إلا أنه يوصى بشدة بعمله كطريقة جيدة للبرمجة العملية . ( وهذا هو نوع تطوير البرنامج المنظم ، والتي صممت لغة البسكال لتشجع على استخدامه ) .
- ٥ - تمثل القيم العددية لنصف القطر والمساحة بالأسماء الرمزية radius , area . وتسمى هذه الأسماء الرمزية بالمتغيرات Variables ، وهي معرفة أو موضحة في السطر الثالث من البرنامج .



٦ - الاسم الرمزي sqrt الموجود في السطر السادس هو دالة قياسية Standard Function تستخدم لحساب مربع نصف القطر .

٧ - لاحظ في النهاية التنقيط الموجود في نهاية كل سطر . فتنتهي معظم الأسطر بفاصلة منقوطة . وبعض الأسطر لا تنتهي بأى تنقيط . كما أن آخر سطر ينتهي بنقطة . وهذا هو جزء من تكوين لغة البسكال .

وسوف نتعرض للتنقيط بتوسع في أقسام لاحقة من أقسام هذا الكتاب .

## ٦ - الخواص الجيدة للبرنامج 6 - DESIRABLE PROGRAM CHARACTERISTICS

دعنا نفحص بعض الخواص الهامة لبرامج الكمبيوتر المكتوبة بطريقة جيدة . وهذه الخواص تنطبق على أى برامج مكتوبة بأى لغة برمجة ، ولاتقتصر على البرامج المكتوبة بلغة البسكال . ويمكنها أن تقدم لنا مجموعة مفيدة من الخطوط الإرشادية فيما بعد في هذا الكتاب عندما نبدأ في كتابة برامج بلغة البسكال .

١ - تشير السلامة integrity إلى دقة الحسابات ، فيجب أن يكون واضحاً أن كل التعزيزات الأخرى للبرنامج لاعمى لها إذا لم تنفذ الحسابات بطريقة صحيحة . وعلى هذا ... فإن سلامة الحسابات هي ضرورة حتمية في أى برنامج من برامج الكمبيوتر .

٢ - ويشير الوضوح Clarity إلى إمكانية القراءة الشاملة للبرنامج ، مع التركيز الخاص على المنطق الذى يحتويه . فإذا ماكتب البرنامج بوضوح ، فيجب أن يكون في مقدرة أى مبرمج آخر أن يتتبع منطق البرنامج ، بدون بذل مجهود . ( كما يجب أن يكون ممكناً لكاتب البرنامج أيضاً أن يتتبع برنامجه بعد تركه للبرنامج فترة زمنية طويلة ) . وأحد أهداف التصميم في لغة البسكال ، هو إعداد برامج واضحة ومقروءة من خلال منهج منظم للبرمجة .

٣ - البساطة Simplicity . عادة ما يتم تحقيق وضوح البرنامج ودقته بجعل الأشياء سهلة بقدر الإمكان ، وجعلها متناسقة مع الأهداف الشاملة للبرنامج . وفى واقع الأمر ، قد يكون مرغوباً فيه التضحية بقدر معين من كفاءة الحسابات ، وذلك للحفاظ على البساطة النسبية ، وعلى هيكل مباشر للبرنامج .

٤ - وتهتم الكفاءة efficiency بسرعة التنفيذ وكفاءة استغلال الذاكرة . وهناك أهداف عامة ، بالرغم من أنه يجب عدم تحقيقها على حساب الوضوح والبساطة . تتطلب معظم البرامج المعقدة المساومة بين هذه الخواص . وفى مثل هذه المواقف تكون الخبرة والإحساس العام من العوامل الأساسية .

٥ - التجزئة modularity . معظم البرامج الكبيرة يمكن تجزئتها إلى عدة أنشطة جزئية محددة . ومن البرمجة العملية الجيدة ، تنفيذ كل من هذه الأنشطة الجزئية كبرنامج لجزء module منفصل . ( ويشار إلى مثل هذه الأجزاء في البسكال بأنها إجراءات procedures أو دوال functions ) . ويمرر استخدام التجزئة في البرمجة من دقة ووضوح البرنامج ، كما أنه يسهل من إجراء أى تعديلات مستقبلية على البرنامج .

٦ - العمومية generality . عادة ما نريد أن يكون البرنامج عاماً بقدر الإمكان ، وداخل حدود معقولة . فمثلاً يمكننا أن نصمم برنامجاً لقراءة قيم مؤشرات parameters رئيسية معينة ، بدلاً من وضع قيم ثابتة في البرنامج . وكقاعدة عامة ... يمكن الحصول على كمية عمومية معتبرة ببذل مجهود إضافي بسيط في البرمجة .

## Review Questions

## أسئلة للمراجعة

- ١ - ماذا يعنى جهاز الكمبيوتر الكبير ؟ وأين يمكن أن توجد مثل هذه الأجهزة ؟ وفى أى شئ تستخدم هذه الأجهزة بصفة عامة ؟
- ٢ - ماهو جهاز المينى كمبيوتر ؟ وماهو اختلاف أجهزة المينى كمبيوتر عن أجهزة الكمبيوتر الكبيرة ؟
- ٣ - ماهو جهاز الميكروكمبيوتر ؟ وماهو اختلاف أجهزة الميكروكمبيوتر عن أجهزة المينى كمبيوتر وأجهزة الكمبيوتر الكبيرة ؟
- ٤ - اذكر نوعين مختلفين من البيانات .
- ٥ - ماذا يعنى برنامج الكمبيوتر ؟
- ٦ - ماذا يحدث بصفة عامة عند تنفيذ برنامج كمبيوتر ؟
- ٧ - ماهى ذاكرة الكمبيوتر ؟ ومانوع المعلومات التى تخزن فيها ؟
- ٨ - ماذا تعنى ( بت ) ؟ وماذا تعنى ( بايت ) ؟
- ٩ - ماهو الفرق بين ( البايت ) و ( الكلمة ) بالنسبة للذاكرة ؟
- ١٠ - ماهى المصطلحات المستخدمة فى وصف ذاكرة الكمبيوتر ؟ وماهى بعض السعات المعتادة ؟
- ١١ - اذكر بعض وحدات الذاكرات الثانوية . وكيف تختلف هذه الذاكرات عن ذاكرة الكمبيوتر الرئيسية ؟
- ١٢ - ماهى وحدات الوقت المستخدمة للتعبير عن سرعة تنفيذ الأنشطة الفردية بواسطة الكمبيوتر ؟
- ١٣ - ماهو الفرق بين تشغيل الدفعة والتشغيل المتداخل ؟ وماهى مميزات وعيوب كل منهما ؟
- ١٤ - ماذا تعنى الشاشة المرئية Console ؟
- ١٥ - ماذا تعنى المشاركة الزمنية ؟ وأى نوع من أنواع التطبيقات تناسبه المشاركة الزمنية ؟
- ١٦ - ماهى لغات الآلة ؟ وماهو الفرق بينهما وبين اللغات مرتفعة المستوى ؟
- ١٧ - اذكر بعض اللغات مرتفعة المستوى شائعة الاستخدام .
- ١٨ - ماهى مميزات استخدام اللغات مرتفعة المستوى ؟
- ١٩ - ماذا تعنى الترجمة ؟ وماذا يعنى التفسير ؟ وكيف تختلف هاتان العمليتان عن بعضهما ؟
- ٢٠ - ماهو برنامج المصدر ؟ وماهو برنامج التشغيل ؟ وماسبب أهمية هذين المفهومين ؟
- ٢١ - ماذا تعنى البرمجة المرتبة ؟
- ٢٢ - من هو ( بليز بسكال ) ؟ ولماذا ذكر اسمه فى هذا الفصل ؟
- ٢٣ - من هو أول من طور لغة البسكال ؟ وماذا كان هدفه الأساسى ؟

- ٢٤ - لخص الحالة الحالية لقياسية لغة البسكال في كل من أوروبا والولايات المتحدة الأمريكية .
- ٢٥ - ماهما المكونان الرئيسيان الذي يجب وجودهما في كل برنامج مكتوب بلغة البسكال ؟ وماهو الغرض من كل منهما ؟
- ٢٦ - ماهما الجزآن الرئيسيان للمجموعة ؟ وماهو الغرض من كل منهما ؟
- ٢٧ - ماهى الكلمات المحجوزة أو الكلمات الرئيسية ؟
- ٢٨ - ماهى الدوال القياسية ؟
- ٢٩ - لماذا ترحل بعض العبارات داخل البرنامج المكتوب بلغة البسكال ؟
- ٣٠ - لخص معنى كل من خواص البرامج التالية : السلامة ، والوضوح ، والبساطة ، والكفاءة ، والتجزئة ، والعمومية .  
ماسبب أهمية كل خاصية من هذه الخواص ؟ .

## Problems

## مشاكل

- ٣١ - حدد الغرض من كل من البرامج المكتوبة بلغة البسكال التالية :

```
(A) PROGRAM greeting(output);
    BEGIN
        writeln('Welcome to the Wonderful');
        writeln;
        writeln('World of Computing!')
    END.

(b) PROGRAM payroll(input,output);
    CONST rate = 0.14;
    VAR gross,tax,net : real;
    BEGIN
        read(gross);
        tax := rate*gross;
        net := gross-tax;
        write(gross,tax,net)
    END.

(c) PROGRAM order(input,output);
    VAR a,b : integer;
    BEGIN
        read(a,b);
        IF a <= b THEN write(a,b) ELSE write (b,a)
    END.
```

- ٣٢ - حدد بالنسبة للبرامج المكتوبة بلغة البسكال في المشكلة السابقة ، الغرض من كل سطر من أسطر كل برنامج من هذه البرامج ، وذلك على قدر استطاعتك .



## الفصل الثانى

### أساسيات البسكال

## Pascal Fundamentals

يهتم هذا الفصل بالعناصر الأساسية المستخدمة في تكوين عبارات بسيطة بلغة البسكال . وتشمل هذه العناصر مجموعة رموز البسكال ، والكلمات المحجوزة ، والمعرفات ، والأرقام ، والسلاسل ، والثوابت ، والمتغيرات والتعبيرات . وسوف نرى كيف يمكن خلط هذه العناصر لتكوين عبارات بسيطة ، لكنها كاملة بلغة البسكال . ويجب توجيه بعض الانتباه أيضا إلى أنواع العبارات المختلفة المتاحة ، والعناصر التي تحتويها ، والأهداف منها .

بعض هذه المادة مشروح بتفصيل كبير ، وعلى هذا ... فقد توجد صعوبة بعض الشيء في استيعابه ، خاصة بالنسبة للمبرمج الذى ليس لديه أى خبرة بالبرمجة . تذكر على أية حال أن الغرض الحالى من هذه المادة هو تقديم بعض المفاهيم الأساسية ، وبعض التعريفات اللازمة للمواضيع التي سوف تذكر في الفصول القليلة التالية . وعلى ذلك ... فعند قراءة هذا الفصل لأول مرة ، فإنك لست في حاجة إلا إلى اكتساب اعتيادا عاما بالمواضيع الفردية . وسوف نكتسب فهما أكثر تفصيلا من الإشارة المتكررة لهذه المادة ، والتي تحدث في الفصول التالية .

### ١ - مجموعة رموز البسكال : 1. THE PASCAL CHARACTER SET

تستخدم لغة البسكال الحروف الهجائية من A إلى Z ( كل من الحروف الكبيرة والحروف الصغيرة ) ، والأرقام من 0 إلى 9 وبعض الرموز الخاصة ، وذلك في بناء مجموعة من العناصر الأساسية للبرنامج ( .الأرقام والمعرفات والتعبيرات وغيرها ) . وفيما يلي توجد الرموز الخاصة :

(	<	.	+
)	<=	:	-
[	>	;	*
]	>=	,	/
{	<>	'	:=
}	..	^	=

وتستخدم الرموز ( \* and \* ) في بعض أجهزة الكمبيوتر ، بدلا من ( and ) . كما يمكن استخدام الرموز ( . and . ) ، بدلا من [ and ] . ويمكن أيضا أن يحل الرمز @ محل ^ في بعض أجهزة الكمبيوتر .

لاحظ أن بعض هذه الرموز مكون من رمزين منفصلين متتاليين ( مثل <= ، = ، و .. الخ ) .

### ٢ - الكلمات المحجوزة : 2. RESERVED WORDS

هناك كلمات محجوزة معينة ، لها معنى قياسي سبق تعريفه في لغة البسكال . وفيما يلي هذه الكلمات :

AND	END	NIL	SET
ARRAY	FILE	NOT	THEN
BEGIN	FOR	OF	TO
CASE	FUNCTION	OR	TYPE
CONST	GOTO	PACKED	UNTIL
DIV	IF	PROCEDURE	VAR
DO	IN	PROGRAM	WHILE
DOWNT0	LABEL	RECORD	WITH
ELSE	MOD	REPEAT	

ويمكن استخدام هذه الكلمات المحجوزة للغرض المحدد لها فقط . ولا يمكن أن يعرفها المبرمج أى تعريف اختياري آخر . ومن المعتاد عرض الكلمات المحجوزة فى برنامج البسكال . إما فى الحروف العلوية ، أو على شكل بارز . وسوف نستخدم الحروف العلوية خلال هذا الكتاب .

### ٣ - المعرفات : 3. IDENTIFIERS

المعرف Identifier هو اسم يعطى لأحد عناصر البرنامج ، مثل : الثابت ، أو المتغير ، أو الإجراء ، أو البرنامج . وتتكون المعرفات من حروف أو أرقام بأى ترتيب ، فيما عدا أول خانة ، والتي يجب أن تحتوى على حرف . ويسمح باستخدام الحروف الكبيرة والحروف الصغيرة . و ليس هناك تمييز للحروف الكبيرة يميزها عن الحروف الصغيرة . وسوف نستخدم على أية حال الحروف الصغيرة فى هذا الكتاب فى المعرفات ، وذلك لتمييزها عن الكلمات المحجوزة ( والتي تظهر بحروف كبيرة ) .

كما تسمح بعض صيغ البسكال ببعض رموز أخرى أيضا ، مثل الشرطة التى توضع تحت الحرف .

مثال (٢-١)

الأسماء التالية هى معرفات صحيحة :

x	yl2	sum	temperature
names	area	taxrate	table1

والأسماء التالية هى معرفات غير صحيحة للأسباب المذكورة أمام كل منها :

c-max لايسمح باستخدام رموز غير الحروف والأرقام .

( فى بعض صيغ البسكال يمكن أن يكون هذا المعرف صحيحاً ) .

4th أول خانة يجب أن يشغلها حرف أبجدي .

array كلمة ARRAY هى كلمة محجوزة .

last word غير مسموح باستخدام فراغ ( تذكر أن الفراغ يعتبر رمزا ) .

ويمكن أن يكون للمعرف أى طول ، إلا أن بعض مترجمات البسكال لا تميز إلا أول 8 خانات فقط . وفى مثل هذه الحالة ، فإن بقية الخانات لا توجد إلا لإظهار معنى معين للمبرمج .

## مثال (٢-٢)

المعرفان filemanager و filemanagement هما معرفان صحيحان من ناحية القواعد . إلا أن الكمبيوتر قد لا يستطيع أن يميز بينهما على أية حال ، بسبب أن أول ثماني خانات تشغلها نفس الحروف في كل من المعرفين . وعلى هذا ... فيجب ألا يستخدم إلا معرف واحد منهما في نفس البرنامج ( إلا إذا كان معروفاً أن مترجم البسكال المستخدم لا يضع قيوداً على طول المعرف ) .

وكقاعدة عامة ... يجب أن يحتوى المعرف على عدد كافٍ من الرموز ، بحيث يكون معناه واضحاً . وفي الناحية الأخرى ، يجب تجنب استخدام عدد كبير من رموز لا معنى لها .

## مثال (٣-٢)

في برنامج بسكال مكتوب لحساب قيمة الاستثمار المستقبلية ، يمكن أن يكون لقيمة الاستثمار والقيمة المستقبلية أسماء رمزية مناسبة . وعلى أية حال ... فاستخدام v و fv ربما يكون موجزاً جداً ، حيث إن الهدف من هذه المعرفات لا يكون واضحاً . وفي الناحية الأخرى ... فإن المعرف futurevalueofaninvestment يكون غير مقنع ، نظراً لأنه طويل ومرهق .

## 4. STANDARD IDENTIFIERS

### ٤ - المعرفات القياسية :

تحتوى لغة البسكال على بعض المعرفات القياسية التي لها معانٍ سبق تحديدها . وهذه المعرفات القياسية هي :

abs	false	pack	sin
arctan	get	page	sqr
boolean	input	pred	sqrt
char	integer	put	succ
chr	ln	read	text
cos	maxint	readln	true
dispose	new	real	trunc
eof	odd	reset	unpack
eoln	ord	rewrite	write
exp	output	round	writeln

وتحتوى بعض صيغ البسكال على معرفات قياسية أخرى إضافية . ويجب على القارئ أن يحدد بدقة المعرفات القياسية المتاحة في صيغة اللغة التي يستخدمها .

وعلى عكس الكلمات المحجوزة ( والتي لا يمكن إعادة تعريفها على الإطلاق ) ، يمكن للمبرمج أن يعيد تعريف المعرفات القياسية . ويتم عمل ذلك باستخدام إيضاحات وتعريفات مناسبة ، كما سيوضح ذلك فيما بعد في هذا الكتاب . ومن البرمجة الجيدة في معظم الأحوال على أية حال أن تستخدم المعرفات القياسية طبقاً للغرض الذي سبق تحديده لها . وهذا صحيح بصفة خاصة بالنسبة للبرامج التي يكتبها المبرمجون المبتدئون . وعلى هذا ... فيجب معاملة المعرفات القياسية بنفس طريقة معاملة الكلمات المحجوزة .

## 5. NUMBERS

### ٥ - الأعداد :

يمكن كتابة الأعداد بعدة طرق مختلفة في لغة البسكال . ويمكن أن يشتمل العدد على إشارة ، أو علامة عشرية ، أو أس ( أو معامل للقوة المرفوع لها الرقم scale factor ) إذا كانت هناك حاجة لذلك . والقواعد التالية تطبق مع كل الأعداد .

- (١) لا يمكن أن يحتوى العدد على فاصلة أو على فراغ .  
 (٢) يمكن أن تسبق العدد إشارة موجب ( + ) ، أو إشارة سالب ( - ) إذا كانت هناك حاجة لذلك . فإذا لم توجد إشارة ، فيعتبر العدد موجبا .  
 (٣) لا يمكن أن يتعدى العدد قيما معينة كحد أدنى وكحد أقصى . وتعتمد هذه القيم على كل من نوع العدد ، ونوع الكمبيوتر المستخدم ، وعلى نوع المترجم المستخدم أيضا .

## الأعداد الصحيحة : Integer Numbers

لاحتوى العدد الصحيح integer number على علامة عشرية ، أو على أس . وعلى هذا ... فإن العدد الصحيح يكون عبارة عن سلسلة من الأرقام ، تسبقها ( بصورة اختيارية ) إشارة موجب أو إشارة سالب .

مثال (٢-٤)

الأعداد الصحيحة التالية هي أعداد صحيحة :

0	1	+1	-1
743	-5280	60000000	-999999

والأعداد الصحيحة التالية غير صحيحة طبقا للأسباب المذكورة أمام كل منها :

123,456 ممنوع استخدام الفواصل في العدد الصحيح .

36. ممنوع استخدام العلامة العشرية في العدد الصحيح .

10 20 30 ممنوع استخدام الفراغات في العدد الصحيح .

ويمكن أن تتراوح قيمة العدد الصحيح من صفر إلى قيمة قصوى تختلف قيمتها من جهاز كمبيوتر لجهاز آخر ، ومن مترجم لمترجم آخر . وهناك قيمة قصوى معتادة لأجهزة الميكروكمبيوتر ، وهي 32767 ، كما أن بعض أجهزة الكمبيوتر الأخرى تسمح بأعداد صحيحة أكبر من هذا العدد كثيرا . ( وتعرف القيمة القصوى باستخدام المعرف القياسي maxint كما هو موضح فيما بعد ) . ويجب أن يحدد القارئ أقصى قيمة متاحة لدى الكمبيوتر الذى يستخدمه .

## الأعداد الحقيقية : Real Numbers

يجب أن يحتوى العدد الحقيقى real number على علامة عشرية ، أو على أس ( أو كليهما ) . فإذا ما استخدمت علامة عشرية مع العدد ، فيجب أن تظهر بين رقمين . وعلى هذا ... فلا يمكن أن يبدأ أو ينتهى العدد الحقيقى بعلامة عشرية .

مثال (٢-٥)

الأعداد الحقيقية التالية هي أعداد صحيحة :

0.0	1.0	-0.2	827.602
50000.0	-0.000743	12.3	-315.0066



والأعداد الحقيقية التالية هي أعداد غير صحيحة ، طبقاً للأسباب المكتوبة أمام كل منها :

1. يجب ألا ينتهى العدد الحقيقى بعلامة عشرية .

1,000.0 ممنوع استخدام الفواصل .

333333. يجب ألا يبدأ العدد الحقيقى بعلامة عشرية .

50 يجب أن تظهر علامة عشرية أو أس فى العدد الحقيقى .

ويمكن أن يحتوى العدد الحقيقى على أس ، وذلك لترحيل موقع العلامة العشرية . ( إذا لم توجد علامة عشرية فى العدد ، فيفترض أنها موجودة على يمين آخر رقم ) . وهذا مثل التمثيل العلمى بالضرورة ، فيما عدا أن الأساس 10 يستبدل بالحرف E أو e . وعلى هذا ... فالعدد  $1.2 \times 10^{-3}$  يكتب بالبسكال على الصورة 1.2E-3 ، أو الصورة 1.2e-3 . ويجب أن يكون الأس نفسه رقماً صحيحاً موجباً أو سالباً .

مثال (٦-٢)

يمكن تمثيل الكمية  $3 \times 10^{10}$  بلغة البسكال فى إحدى الصيغ التالية

3.0E+10	3.0E10	3e+10	3E10
0.3E+11	0.3e11	30.0E+9	30e9

وبالمثل يمكن تمثيل الكمية  $-5.026 \times 10^{-17}$  بلغة البسكال بإحدى الصيغ التالية :

-5.026E-17	-0.5026E-16	-50.26e-18	-0.0005026e-13
------------	-------------	------------	----------------

مثال (٧-٢)

الأعداد الحقيقية التالية هي أعداد صحيحة :

2E-8	-0.006e-5	1.6667E+8	+0.12121212e12
------	-----------	-----------	----------------

والأعداد الحقيقية التالية خاطئة طبقاً للأسباب المذكورة أمام كل منها :

3.E+10 يجب أن يظهر رقم فى كل ناحية من نواحي العلامة العشرية .

8e2.3 الأس يجب أن يكون صحيحاً .

3333e-3 يجب أن يظهر رقم قبل العلامة العشرية .

3E 10 ممنوع ظهور الفراغات فى الرقم .

والأعداد الحقيقة لها مدى أكبر كثيراً من الأعداد الصحيحة . وعادة ماتتروح قيمة الأعداد الحقيقية من أقل قيمة تعادل 1E-38 إلى أقصى قيمة تعادل 1E+38 . ( وتتغير هذه القيم من جهاز كمبيوتر لجهاز آخر ، ومن مترجم

بسكال لترجم آخر ) . وبالإضافة إلى ذلك ... فإن الرقم 0.0 ، وهو أقل من 1E-38 ، هو رقم صحيح أيضا . ويجب أن يحدد القارئ القيم المناسبة للجهاز المتاح له .

ويتغير عدد الأرقام المعنوية في العدد الحقيقي من إحدى صيغ لغة البسكال لصيغة أخرى . وتسمح معظم الصيغ بسبعة أو ثمانية أرقام معنوية ، والتي تكفى بالنسبة لمعظم التطبيقات ، ويجب أن يحدد القارئ عدد الأرقام الصحيحة المعنوية المناسبة للجهاز الذي يستخدمه .

## Numerical Precision

## الدقة العددية :

يجب أن يكون مفهوما أن الأعداد الصحيحة هي كميات مضبوطة ، بينما الأعداد الحقيقية هي تقريبات . وعلى هذا ... فإن العدد الحقيقي 1.0 يمكن أن يكون ممثلا داخل ذاكرة الكمبيوتر على أنه .....0.99999999 ، بالرغم من أنه يمكن أن يظهر على أنه 1.0 على الشاشة ، أو إذا ما تمت طباعته . وعلى هذا ... فلا يمكن استخدام الأعداد الحقيقية في أغراض خاصة ، مثل العد ، أو عمل الفهارس ، وما إلى ذلك ، حيث تكون القيم الدقيقة مطلوبة . وسوف نناقش هذه القيود كلما دعت الحاجة لذلك في فصول أخرى من هذا الكتاب .

## 6. STRINGS

## ٦ - السلاسل :

السلسلة string هي تسلسل من الرموز ( مثل الحروف والأرقام والرموز الخاصة ) ، وتكون موضوعة بين علامتي تنصيص .

### مثال (٢-٨)

السلاسل التالية هي سلاسل صحيحة :

'GREEN'	'Washington, D.C. 20005'	'270-32-3456'
'\$19.95'	'THE CORRECT ANSWER IS:'	'2*(I+3)/J'

وأقصى عدد للرموز التي يمكن أن توجد بين علامتي التنصيص يتغير من إحدى صيغ البسكال لصيغة أخرى . ومعظم الصيغ تسمح بأن يكون أقصى طول للسلسلة 256 رمزا . وهو طول كافٍ لمعظم الأغراض .

وإذا ما احتوت السلسلة على علامتي تنصيص ، فيجب أن تمثل علامة التنصيص بعلامتي تنصيص متتاليتين . فإذا ما حدث ذلك ، فلن يظهر في الطباعة سوى علامة تنصيص واحدة . وعلى ذلك ... فتفسر علامة التنصيص الواحدة بأنها محدد للسلسلة string delimiter ، بينما تفسر علامتا التنصيص المتتاليتان كعلامة تنصيص موجودة داخل السلسلة .

### مثال (٢-٩)

يمكن أن يحتوى برنامج بسكال على السلسلة التالية :

'PLEASE DON'T VERB YOUR NOUNS'

( لاحظ علامة التنصيص المتكررة في الكلمة "don't" ) ، فإذا ما طبع البرنامج هذه السلسلة ، فإنها تظهر على النحو التالي :

PLEASE DON'T VERB YOUR NOUNS

وعادة ما تستخدم السلاسل في كتابة عبارات كعناوين للمخرجات . وسوف نناقش ذلك بالتفصيل في الفصل الرابع من الكتاب .

## ٧ - أنواع البيانات :

### 7. DATA TYPES

أحد الخواص الهامة والممتعة للغة البسكال هو مقدرتها على دعم العديد من أنواع البيانات المختلفة . ويشمل هذا بيانات من نوع بسيط ، وبيانات من نوع مركب ، وبيانات من نوع مشير .

والبيانات من النوع البسيط Simple-type data هي عناصر فردية ( أعداد أو رموز .. الخ ) ، وتصاحبها معرفات فردية على أساس معرف لكل منها . وفي واقع الأمر ... فإنه هناك أنواع عديدة من البيانات البسيطة . وهذه تشمل الأربعة أنواع القياسية للبيانات ، وهي : بيانات صحيحة ، وبيانات حقيقية ، وبيانات حرفية ، وبيانات بوليان ، والأنواع البسيطة التي يعرفها المستفيد والتي تشمل النوع المتعدد ونوع المدى الجزئي . وأنواع البيانات القياسية سوف نتناقص بالتفصيل في الفصل القادم . أما الأنواع البسيطة التي يعرفها المستفيد ، فسوف تؤخذ في الاعتبار في الفصل الثامن من الكتاب .

والبيانات من النوع المرتب structured-type data تحتوي على عناصر بيانات متعددة ، وترتبط عناصرها بعنصر بنفس الطريقة المحددة لذلك . ويصاحب كل مجموعة من عناصر البيانات معرف خاص . ويمكن أن يصاحب كل عنصر بيان فردي موجود داخل معرف خاص مناظر له . وهناك أربعة أنواع من البيانات المرتبة في البسكال ، وهي المنظومات والسجلات والملفات والفئات .

والبيانات من النوع المشير pointer-type data تستخدم لتكوين بيانات ديناميكية من النوع المركب . والوصف البسيط لخواصها واستخداماتها يقع في مدى خارج المناقشة الحالية ( انظر الفصل الثالث عشر من الكتاب ) .

وأنواع البيانات المختلفة ملخصة أدناه :

#### ( أ ) بيانات بسيطة :

##### ( ١ ) بيانات قياسية :

- ( ١ ) صحيح .
- ( ٢ ) حقيقي .
- ( ٣ ) حرفي .
- ( ٤ ) بوليان .

##### ( ب ) بيانات يعرفها المستفيد :

- ( ١ ) متعددة .
- ( ٢ ) مدى جزئي .

(٢) بيانات هرتبة :

(أ) منظومات .

(ب) سجلات .

(ج) ملفات .

(د) فئات .

(٣) بيانات مشيرة .

وسوف نهتم حاليا بالبيانات البسيطة فقط ، حيث ستناقش البيانات المركبة والبيانات المشيرة في فصول لاحقة من الكتاب .

## 8. CONSTANTS

٨ - الثوابت :

من المقنع عادة تتبع عنصر بيانات بسيط ، مثل القيمة العددية أو السلسلة ، والذي يكون له معرف ، والذي يقدم اسماً لعنصر البيانات . ويسمى المعرف ثابتاً constant إذا ما حُد عنصر البيانات بصورة دائمة ( أى إذا كانت قيمة عنصر البيانات غير متغيرة طوال فترة تنفيذ البرنامج )

ويجب أن يعرف الثابت دائماً قبل أن يظهر في أى عبارة من عبارات البسكال . ويخدم هذا التعريف هدفين ، حيث إنه يحدد أن المشير ثابت ، كما أنه يصاحب قيمة الثابت . وسوف يقوم عنصر البيانات بتحديد الثابت داخلياً .

والصيغة العامة لتعريف الثابت تكتب على النحو التالي :

CONST name = value

حيث name هو معرف يمثل اسم الثابت ، و Value هي عنصر البيانات الفعلى الذي يحدد له المعرف name .

مثال (٢-١٠)

تطلب أحد برامج البسكال استخداماً متكرراً للقيمة العددية 0.1666667. وعلى هذا ... فقد يكون من المقنع إدخال ثابت اسمه fraction ليستخدم في مكان القيمة العددية الحقيقية . ويمكن تعريف هذا الثابت بكتابة مايلي :

CONST fraction = 0.1666667;

والإشارة التالية للمعرف fraction تكون مكافئة للإشارة إلى القيمة العددية الحقيقية . لاحظ أن fraction يعتبر ثابتاً حقيقياً ، حيث إنه يصاحبه عدد حقيقى .

مثال (٢-١١)

افترض أن أحد برامج البسكال يتكرر فيه استخدام السلسلة التالية :

'The Super-Duper Computer Company'

وذلك في كتابة عناوين بأحد التقارير . ويمكن تمثيل هذه السلسلة بسهولة كثابت اسمه title . ولعمل ذلك ، فإننا نكتب مايلي :

CONST title = 'The Super-Duper Computer Company';

وذلك في بداية البرنامج ... فإذا ما أردنا أن نطبع السلسلة الفعلية في نقطة لاحقة من نقاط البرنامج ، فإننا نشير إلى المعرف title في عبارة المخرجات المناسبة . ( سوف نناقش عمليات المدخلات والمخرجات في الفصل الرابع من الكتاب ) .

ويعتبر title في هذا المثال ثابتاً من نوع السلسلة ، حيث إنه هناك سلسلة مصاحبة له .

## 9. VARIABLES

### ٩ - المتغيرات :

المعرف الذي يسمح لقيمته بالتغير أثناء تنفيذ البرنامج يسمى متغيراً Variable . ويجب توضيح كل متغير على حدة ، أي يجب تعريف كل متغير على حدة ، وذلك قبل أن يظهر في البرنامج . ويحدد توضيح المتغير الحقيقي بأن المعرف هو متغير ( بدلاً من أن يكون ثابتاً أو غير ذلك ) كما يحدد نوع المتغير . وعلى عكس تعريف الثابت ، فلا يكون هناك عنصر بيانات ( أي قيمة عددية أو سلسلة ) مصاحبة للمتغير ، وذلك في توضيح أو تعريف المتغير .

والصيغة العامة لتوضيح المتغير هي :

VAR name : type

أو إذا كان هناك متغيرات متعددة من نفس النوع ، فإن الصيغة العامة تصبح :

VAR name 1, name 2, . . . , name n : type

حيث إن name 1 و name 2 .. الخ هي معرفات تمثل أسماء فردية لمتغيرات . وتشير type إلى نوع بيانات المتغير .

### مثال (٢-١٢)

يحتوي أحد برامج البسكال على متغيرات صحيحة ، اسمها row و column ، ومتغير حقيقي اسمه value ، ومتغير حرفي اسمه flag . وعلى هذا ... فيمكن أن يحتوي البرنامج على التوضيحات التالية :

```
VAR row,column : integer;
    value : real;
    flag : char;
```

وعنصر البيانات المختار يمكن أن يحدد بأنه ينتمي إلى أحد هذه المتغيرات ، وذلك فيما بعد في البرنامج .

### مثال (٢-١٣)

افترض أن البرنامج يحتوي الآن على الثوابت الموصوفة في المثال رقم ٢ - ١٠ ، والمثال رقم ٢ - ١١ ، والمتغيرات الموصوفة في المثال رقم ٢ - ١٢ . في هذه الحالة تصبح قائمة التوضيحات الكاملة على النحو التالي :

```
CONST fraction = 0.1666667;
    title = 'The Super-Duper Computer Company';
VAR row,column : integer;
    value : real;
    flag : char;
```

تذكر أنه يجب أن تظهر التوضيحات والتعريفات في ترتيب معين ، وأنه يجب أن تسبق تعريفات الثوابت لتوضيحات المتغيرات .

## 10. EXPRESSIONS

### ١٠ - التعبيرات :

التعبير expression هو عبارة عن تجميع من العناصر لعناصر operands ( أى الأرقام أو الثوابت أو المتغيرات .. الخ ) متصلة مع بعضها بواسطة مؤثرات Operators لتكوين اصطلاح جبرى يمثل قيمة ( مثل عنصر بيانات بسيط ) . وهناك نوعان من التعبيرات فى البسكال ، وهى : تعبيرات عددية ، وتعبيرات بوليانية . والتعبير العددى numerical expression يمثل قيمة عددية ، بينما يمثل التعبير البوليانى boolean expression شرطا منطقيا ، تكون قيمته صحيحة أو خاطئة فقط .

#### مثال (٢-١٤)

فيما يلى مثالا لتعبير عددى

$$(b*b-4*a*c)/(2*a)$$

وتسمى المعرفات a و b و c والأرقام 4 و 2 بأنها العناصر التى يؤثر عليها operands ، بينما تسمى الرموز \* و - و / بأنها المؤثرات operators المناظرة ( التى تمثل الضرب والطرح والقسمة على التوالي ) . وتستخدم الأقواس لتحديد ترتيب تنفيذ العمليات . ويمثل التعبير عددا معينا ، وعلى هذا ... فإذا ما كانت a و b و c تمثل القيم 1 و 2 و 3 على التوالي ، فإن هذا يعنى أن التعبير يمثل القيمة 4 .

وعند تكوين تعبير عددى ، يجب التمييز بين الكميات الصحيحة والكميات الحقيقية . وهذا صحيح بالنسبة لكل من العناصر التى يؤثر عليها ، وكذلك التعبير نفسه . وسوف نتعرض للمزيد عن ذلك فى الفصل التالى .

#### مثال (٢-١٥)

فيما يلى مثالا لتعبير بوليانى

$$\text{pay} < 1000.0$$

فى هذا التعبير يكون pay متغيرا من النوع الحقيقى ، حيث إن 1000.0 هى عدد حقيقى ، والرمز < هو مؤثر بوليانى . لاحظ أن كل من pay و 1000.0 هى عناصر يؤثر عليها فى هذا التعبير البوليانى ( . وتكون قيمة التعبير صحيحة إذا مامتت pay قيمة أقل من 1000.0 . وعلى العكس من ذلك ... تصبح قيمة التعبير خاطئة إذا مامتت pay قيمة أكبر من 1000.0 أو تساويها .

وتستخدم التعبيرات البوليانية فى العديد من تكوينات التحكم ، مثل مكون IF - THEN التالى :

```
IF pay < 1000.0 THEN writeln(employeeenumber);
```

وسوف يتسبب هذا المكون فى كتابة قيمة المتغير employeeenumber إذا ماكانت قيمة pay أقل من 1000.0 .

وسوف نناقش استخدام تعبيرات بوليانية داخل تكوينات التحكم بتفاصيل أكثر فى الفصل السادس من الكتاب .

ويجب على كل التعبيرات أن تتبع الشروط العامة التالية :

(١) غير مسموح بكتابة مؤثرين تالين لبعضهما مباشرة . وعلى أية حال ... يمكن استخدام الأقواس لتفصل بين أى مؤثرين تالين . ( تذكر أنه يجب أن تستخدم الأقواس أزواجا أزواجا بصفة دائمة ) .

(٢) يمكن أن يحتوى التعبير على معرف واحد ، يستخدم كثابت أو كمتغير .

(٢) اسم الدالة ( أى دليل الدالة ) يمكن استخدامه فى مكان معرف لثابت ، أو فى مكان معرف لتفسير داخل التعبير ( سوف يذكر المزيد عن ذلك فيما بعد فى هذا الفصل ) .

## 11. STATEMENTS

### ١١ - العبارات :

عبارة statement البسكال هى عبارة عن أحد التعليمات ، أو مجموعة من التعليمات التى تجعل الكمبيوتر يتخذ إجراءً معيناً . وهناك نوعان أساسيان من العبارات فى البسكال : وهما العبارات البسيطة ، والعبارات المرتبة والعبارات البسيطة simple هى عبارات فردية بالضرورة ، وهى تعليمات غير شرطية تنفذ أحد الأنشطة التالية :

(١) تحدد عنصر بيانات لأحد المتغيرات ( وتسمى هذه العبارة بعبارة تحديد assignment ) .

(٢) توصل إجراء حسابى ذاتياً ، ويسمى إجراء procedure .

(٣) تنقل التحكم فى البرنامج غير المشروط إلى جزء آخر من أجزاء البرنامج ( عبارة GOTO ) .

### مثال (١٦-٢)

فيما يلى عبارة تحديد

tax := 0.14\*gross;

فى هذا المثال يفترض أن tax و gross متغيران من النوع الحقيقى ، وأن القيمة الحقيقية قد تحددت للمتغير gross . ( يمكن أن تكون هذه القيمة سبق قراءتها فى الكمبيوتر ، أو أنه سبق حسابها فى البرنامج ) . وتتسبب عبارة التحديد فى ضرب قيمة gross فى 0.14 ، وتحديد أن حاصل الضرب هو قيمة للمتغير tax .

لاحظ أن الرمز المستخدم للتحديد هو ( = : ) وليس = كما فى حالة معظم لغات البرمجة الأخرى . لاحظ أيضاً أن طرف العبارة الأيمن ( وهو gross\*0.14 ) عبارة عن تعبير عددى كما سبق وصف التعبير العددى .

### مثال (١٧-٢)

فيما يلى عبارة من عبارات GOTO

GOTO 100;

ولايشجع استخدام عبارات GOTO فى البسكال . وسوف نذكر الكثير عن ذلك فى الفصل السادس من الكتاب .

سوف نذكر الكثير عن العبارات البسيطة التى تصل الإجراءات بعضها ببعض ، وذلك فى القسم التالى .

وتميز لغة البسكال أنواعاً عديدة من العبارات المرتبة . وهى تشمل مايلى :

(١) عبارات مركبة تحتوى على تسلسل من عبارتين أو أكثر ، متتالية وراء بعضها .

(٢) عبارات متكررة تشمل إعادة تنفيذ لعدة عبارات بسيطة .

(٣) عبارات شرطية تنفذ عبارة بسيطة واحدة أو أكثر ، إذا ماتحقق حدوث شرط منطقى محدد فقط .

### مثال (٢-١٨)

فيما يلي عبارة مركبة مأخوذة من مثال رقم ١ - ٧ من الفصل الأول .

```
BEGIN
  read(radius);
  area := 3.14159*sqr(radius);
  write(radius,area)
END
```

لاحظ أن العبارات البسيطة التي تتكون منها العبارة المركبة محصورة بين الكلمتين BEGIN و END . لاحظ أيضا أن العبارات البسيطة مفصولة عن بعضها باستخدام فواصل منقوطة .

### مثال (٢-١٩)

فيما يلي عبارة متكررة :

```
FOR count := 1 TO 100 DO write(count);
```

سوف تنفذ هذه العبارة 100 مرة . وفي كل مرة تنفذ فيها العبارة تطبع قيمة المتغير Count ، أو تظهر على الشاشة . وعلى هذا ... تتسبب العبارة في ظهور القيم التالية على وحدة المخرجات :

1 2 3 . . . 100

### مثال (٢-٢٠)

فيما يلي عبارة شرطية :

```
IF pay < 1000.0 THEN write('group 1') ELSE write('group 2');
```

تتسبب هذه العبارة في ظهور group 1 في صورة المخرجات إذا ما كان المتغير pay يمثل قيمة أقل من 1000.0 . أما إذا كانت قيمة pay أكبر من 1000.0 أو تساويها . فتظهر group 2 . بدلا من ظهور group 1 .

وسوف تناقش العبارات المرتبة بتفاصيل أكبر في الفصل السادس من الكتاب . ويجب أن يهتم القارئ الآن بالنظرة العامة على المفاهيم العامة فقط .

## ١٢ - الإجراءات والدوال : 12.PROCEDURES AND FUNCTIONS

الإجراءات والدوال هي عناصر محتواة ذاتيا في البرنامج ، وأحيانا ما يشار إليها بأنها أجزاء modules تنفذ إجراءات محددة . ويمكن الاتصال بهذه الإجراءات من أي مكان في البرنامج . فإذا ما تم الاتصال بنفس الإجراءات من عدة نقاط مختلفة في البرنامج ، فيمكن إعطاء هذا الإجراء معلومات مختلفة ( أي قيم مختلفة لعناصر البيانات المطلوبة ) عند كل نقطة من نقاط الاتصال به .

وعند الاتصال بأحد الإجراءات ، فإن المعلومات المقدمة يتم تشغيلها بواسطة العبارات الموجودة في الإجراء . وعادة ما يتسبب ذلك في إنتاج معلومات جديدة . وعند ذلك تسود المعلومات إلى النقطة التي حدث اتصال فيها آخر مرة بالإجراء ، ويستمر البرنامج في التنفيذ عند هذه النقطة .

وتقدم المعلومات التي تمر إلى الإجراء كقائمة بعناصر البيانات ( أي كتوابت ومتغيرات وتعابير .. الخ ) ، والتي تسمى بالمؤشرات parameters . وتفصل هذه المؤشرات عن بعضها بواسطة فواصل ، كما أنها توضع كلها بين قوسين ، وذلك بعد اسم الإجراء مباشرة . ويمكن استخدام مؤشرات محددة في تمثيل معلومات جديدة يتم إنتاجها داخل الإجراء . وعلى هذا ... فيمكن أن تمثل المؤشرات معلومات يعيدها الإجراء أو معلومات تقدم للإجراء .



وتدعم لغة البسكال كل من الإجراءات والدوال القياسية التي يعرفها المستفيد . وسوف نركز على الإجراءات والدوال القياسية حالياً . وعناصر البرنامج هذه تكون موجودة في مكتبة البسكال ، وهي جزء من اللغة . وسوف تقدم مناقشة كاملة للإجراءات والدوال التي يعدها المستفيد في الفصل السابع من الكتاب .

وجميع الإجراءات procedures لها الخواص العامة التالية :

- (١) يتم الوصول إلى الإجراء عن طريق عبارة بسيطة تحتوي على اسم الإجراء يتبعه قائمة ( اختيارية ) بالمؤشرات .
  - (٢) يمكن أن تمثل المؤشرات معلومات تقدم إلى الإجراء ، أو تحت ظروف معينة يمكن أن تمثل معلومات تعود من الإجراء .
  - (٣) يمكن نقل أى عدد من عناصر البيانات بين الإجراء والنقاط التي تشير إليه ( أى العبارة التي تصل إلى الإجراء ) .
- مثال (٢-٢١)

عادة ماتكون عبارة write دليلاً ( أى تصل إلى إجراء ) لإجراء بسكال قياسي . وعلى هذا فإن العبارة التالية :

```
write(a,b,c);
```

سبب في طباعة قيم المؤشرات a و b و c ، أو في إظهارها على الشاشة .

لاحظ أن كل المؤشرات في هذا المثال معلومات تقدم إلى الإجراء ، ولا توجد أى معلومات جديدة تعود منه .

افرض أن برنامج البسكال يحتوي على عبارتي الكتابة write التاليتين :

```
write(a,b,c);
```

```
.
```

```
.
```

```
write(x,y,z);
```

تتسبب أول عبارة في طباعة قيم المتغيرات a و b و c ، بينما تقوم العبارة الثانية ( والمتطابقة تماماً مع العبارة الأولى باستثناء المؤشرات ) بإخراج قيم المتغيرات x و y و z .

وجميع الدوال functions لها الخواص العامة التالية :

- (١) يتم الاتصال بالدالة عن طريق تحديد اسمها في أحد التعبيرات ، كما لو كانت متغيراً عادياً ، ويتبع اسمها قائمة ( اختيارية ) من المؤشرات .
- (٢) يمكن أن يتبع اسم الدالة قائمة ( اختيارية ) من المؤشرات . وتستخدم هذه المؤشرات في نقل معلومات إلى الدالة من نقطة الإشارة إليها فقط .
- (٣) تقوم الدالة بإعادة عنصر بيانات واحد . وسوف يقوم اسم الدالة نفسه بتمثيل عنصر البيانات هذا .
- (٤) يجب أن تكون الدالة من نفس نوع البيانات للتعبير الذي يقوم بالاتصال بها .

### مثال (٢-٢٢)

اعتبر مرة أخرى العبارة التالية :

```
area := 3.14159*sqr(radius);
```

الطرف الأيمن عبارة عن تعبير عددي ، تمر من خلاله قيمة المتغير radius ، وذلك إلى الدالة القياسية sqr . وتعود هذه الدالة بقيمة مربع نصف القطر . وعند ذلك تضرب قيمة مربع نصف القطر في 3.14159 ، وتحدد النتيجة للمتغير area .

لاحظ أن radius هو مؤشر يمثل عنصر بيانات يقدم إلى الدالة القياسية . أما عنصر البيانات الذي يعود من الدالة ، فيمثله اسم الدالة sqr . لاحظ أيضا أن نوع الدالة عددي ( حقيقية بصفة محددة ) ، وهذا ضروري إذا ما كانت الدالة تستخدم في تعبير عددي .

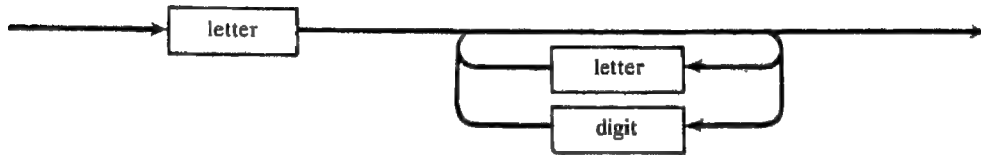
وسوف نذكر المزيد عن الإجراءات والوال في الفصل السابع من الكتاب . ولانحتاج حاليا إلا إلى الاهتمام بالمعلومات العامة التي سبق ذكرها عن هذا الموضوع . وسوف تسمح لنا هذه المعلومات بفهم المادة المعروضة في الفصول القادمة .

### ١٣ - الرسومات التكوينية في لغة البسكال : 13. PASCAL SYNTAX DIAGRAMS

من المناسب قبل ترك هذا الفصل أن تناقش طريقة شائعة الاستخدام في تمثيل الأشكال التكوينية Syntactical ( أى الخاصة بقواعد اللغة ) في لغة البسكال . وهي تشمل استخدام رسومات تكوينية ، مثل الرسم الموجود في المثال التالي :

### مثال (٢-٢٣)

يوضح الشكل ٢ - ١ رسما تكوينيا من الرسومات التكوينية للغة البسكال ، والذي يوضح الطريقة التي يمكن تمثيل المعرف بها . ويبين هذا الشكل أن المعرف يجب أن يبدأ بحرف كما هو موضح في المستطيل الموجود على أقصى يسار الشكل ، والموجود به كلمة letter . ويلى هذا المستطيل مسار مستقيم له نورتان اختياريتان للعودة . تحتوى كل نورة على مستطيل يمثل نوع الرموز التي يمكن تواجدها كجزء من أجزاء اسم المعرف . وحيث إن المستطيل العلوى يحتوى على كلمة letter ، والمستطيل السفلى يحتوى على كلمة digit ، فإننا نستخلص من ذلك أنه يجب أن يبدأ أى معرف بحرف ، ويتبعه أى عدد من الحروف والأرقام .



الشكل ٢ - ١

ويمكن أن تصبح الرسومات التكوينية هذه معقدة جدا ، حتى بالنسبة للتكوينات البسيطة . وعلى هذا ... فمعظم المبتدئين يفضلون عدم استخدامها . وبعد أن يكتسب القارئ بعض الفهم والاعتیاد على لغة البسكال ، فإن مثل هذه الرسومات تبدو أكثر فهما له .

توجد قائمة كاملة بكل الرسومات التكوينية في لغة البسكال في ملحق F . ونوصى القارئ بأن يعود إلى هذه الرسومات بصفة متكررة .

## Review Questions

## أسئلة للمراجعة :

- (١) ماهى محتويات فئة رموز البسكال ؟
- (٢) ماهى الكلمات المحجوزة فى البسكال ؟ وكيف يمكن تعريف الكلمات المحجوزة داخل هذا الكتاب ؟ .
- (٣) ماهى معرفات البسكال ؟ وماهى مكونات المعرف ؟
- (٤) كيف يمكن أن يحتوى المعرف على العديد من الرموز ؟ وهل كل هذه الرموز متساوية المعنى ؟
- (٥) ماذا يعنى المعرف القياسى ؟
- (٦) ماهو الفرق بين المفردات القياسية والكلمات المحجوزة ؟
- (٧) لخص القواعد العامة لكتابة الأعداد فى البسكال .
- (٨) ماهى القواعد الخاصة بالأعداد الصحيحة ؟
- (٩) كيف يمكن تحديد أكبر عدد صحيح مسموح به لكل كمبيوتر ؟
- (١٠) أذكر طريقتين مختلفتين لكتابة الأعداد الحقيقية . لخص القواعد التى تطبق فى كل حالة من هاتين الحالتين .
- (١١) ماهو الغرض من الأس الموجود فى العدد الحقيقى ؟
- (١٢) اذكر الفرق بين الأعداد الحقيقية والأعداد الصحيحة . تحت أى ظروف يجب استخدام كل نوع من هذين النوعين للأعداد ؟
- (١٣) ماهى السلسلة ؟ وماهى الرموز التى يمكن أن تظهر فى السلسلة ؟
- (١٤) ماهى القيود الموضوعة على أقصى طول للسلسلة ؟
- (١٥) كيف يمكن طباعة علامة تنصيص داخل إحدى السلاسل ؟
- (١٦) اذكر ثلاثة أسماء مختلفة للبيانات المستخدمة فى البسكال .
- (١٧) مامعنى البيانات بسيطة النوع ؟
- (١٨) ماهى الأربعة أنواع القياسية للبيانات البسيطة ؟
- (١٩) مامعنى النوع البسيط الذى يعرفه المستفيد ؟ وماهو الفرق بين هذا النوع من البيانات وأنواع البيانات القياسية ؟
- (٢٠) مامعنى البيانات مرتبة النوع ؟ وماهو الفرق بينها وبين البيانات بسيطة النوع ؟
- (٢١) اذكر أربعة أنواع مختلفة من البيانات المرتبة .
- (٢٢) مامعنى الثابت ؟ وماذا يجب أن يحدث قبل أن يظهر الثابت فى عبارة بسكال ؟
- (٢٣) كيف يحدد نوع الثابت ؟

- (٢٤) مامعنى المتغير ؟ وماهو الفرق بينه وبين الثابت ؟
- (٢٥) لخص القواعد المستخدمة فى توضيح أنواع البيانات للمتغيرات .
- (٢٦) ماهو التعبير ؟ وماهى التعبيرات المستخدمة فى البسكال ؟ وماهو الاختلاف بينها ؟
- (٢٧) ماهو العنصر المؤثر عليه ؟ وماهو المؤثر ؟ وكيف يستخدم كل منهما فى التعبير ؟
- (٢٨) لخص الشروط العامة التى يجب أن تحققها كل التعبيرات .
- (٢٩) ماهى العبارة ؟ وماهما النوعان الأساسيان للعبارات فى البسكال ؟ وماهو الفرق بينهما ؟ .
- (٣٠) اذكر ثلاثة أنشطة مختلفة يمكن أن تؤديها العبارة البسيطة ؟
- (٣١) ماهو الغرض من عبارة التحديد ؟ وماهو نوع هذه العبارة ؟
- (٣٢) اذكر ثلاثة أنواع مختلفة للعبارات المرتبة .
- (٣٣) ماهو الإجراء ؟ وماهى الدالة ؟ وماهو الفرق بين كل منهما ؟ .
- (٣٤) ماهو الجزء module ؟ وماهى العلاقة بينه وبين الإجراء procedure والدالة function ؟
- (٣٥) ماذا يحدث عند الاتصال بإجراء ؟
- (٣٦) مامعنى مؤشرات ؟ وماهو الغرض منها ؟
- (٣٧) اذكر فئتين عامتين للإجراءات والدوال .
- (٣٨) لخص الخواص العامة للإجراءات .
- (٣٩) لخص الخواص العامة للدوال .
- (٤٠) ماهى دلالة اسم الدالة ؟ وماهو الفرق بينها وبين اسم الإجراء ؟
- (٤١) مامعنى الشكل التكويني ؟ وكيف تمثل هذه الأشكال فى البسكال ؟

## Solved Problems

## مسائل محلولة :

(٤٢) حدد : أى من المعرفات التالية صحيح . وإذا كان خاطئاً ؛ حدد سبب الخطأ :

record-number خطأ – ممنوع ظهور أى شئ خلاف الأرقام والحروف .

identifier 1 صحيح

identifier 2 صحيح بالرغم من أنه قد لايمكن تمييزه عن identifier 1 ، نظراً لأن طوله يزيد عن 8 خانات والثمانى خانات الأولى هى نفسها فى كل من الإسمين .

1 pointer ممنوع أن يبدأ الاسم برقم .

- first record ممنوع وجود فراغ داخل الاسم .
- to TO هي كلمة محجوزة .
- total صحيح .
- (٤٣) حدد : أى الأعداد التالية صحيح . وإذا كان خاطئاً ؛ حدد سبب الخطأ .
- 666 صحيح وهو عدد صحيح .
- 666. خطأ - يجب أن يظهر رقم على كل جانب من جانبي العلامة العشرية .
- 666.0 صحيح وهو عدد حقيقي .
- 6,66e2 صحيح وهو عدد حقيقي .
- 483,500 خطأ - ممنوع استخدام الفواصل .
- 6 166667e خطأ - ممنوع ظهور الفراغ .
- (٤٤) حدد : أى من السلاسل التالية صحيح :
- 'Visit beautiful, sunny pittsburgh!' صحيح .
- 'The price is \$56.50' صحيح .
- 'It's terribly cold out here!' خطأ لوجود علامة تنصيص واحدة داخل السلسلة .
- (٤٥) حدد كيف يمكن للمعرفات التالية أن تصاحبها القيم الثابتة المناظرة لكل منها .

المعرف	قيمة الثابت
count	3
offset	-2
fraction	0.333333
color	blue

```
CONST count = 3;
      offset = -2;
      fraction = 0.333333;
      color = 'blue';
```

ملاحظة : لاحظ أن هذه هي تعريفات definitions لثوابت .

(٤٦) حدد كيف يمكن للمعرفات التالية أن تصاحبها أنواع البيانات المناظرة لها .

المعرف	نوع البيانات
index	صحيح
cmax	حقيقي
cmin	حقيقي
code	حرفي
status	بوليان
<pre>VAR index : integer;     cmax,cmin : real;     code : char;     status : boolean;</pre>	

ملاحظة : لاحظ أن هذه هي توضيحات declarations للمتغيرات .

(٤٧) عرف نوع كل من التعبيرات التالية :

- |                      |                |
|----------------------|----------------|
| (a) $2*x+7$          | عددي           |
| (b) $count \geq 100$ | بوليان         |
| (c) $sqr(value+3)/5$ | عددي           |
| (d) $value = 666$    | بوليان         |
| (e) test             | عددي أو بوليان |

طبقا لنوع البيانات

المصاحبة للاختبار

(٤٨) فيما يلي عدة عبارات بسكال . عرف أى منها بسيط ، وأى منها مرتب .

- |   |                          |
|---|--------------------------|
| (a) <code>area := length*width;</code>  | بسيطة ( عبارة تحديد )    |
| (b) <code>IF count = 100 THEN write(a,b,c);</code>  | مرتبة ( شرطية )          |
| (c) <code>GOTO 200;</code>  | بسيطة ( تحويل غير شرطي ) |
| (d) <code>read(length,width);</code>  | بسيطة ( اتصال بإجراء )   |
| (e) <code>BEGIN</code><br><code>read(length,width);</code><br><code>area := length*width;</code><br><code>write(length,width,area)</code><br><code>END</code> | مرتبة ( مركبة )          |
| (f) <code>FOR index := 100 DOWNT0 1 DO write(index);</code>   | مرتبة ( تكرارية )        |
| (g) <code>mean := sqrt(sqr(a)+sqr(b)+sqr(c));</code>  | بسيطة ( تحديد )          |

(٤٩) أى عبارة من عبارات السؤال السابق تتصل بإجراء ؟ وأى منها يحتوى على إشارات لدالة ؟

يمكن الاتصال بإجراءات باستخدام العبارات b , d , e , f . ويشار إلى دالة ( أى يتم الاتصال بها ) بأخر عبارة فقط من عبارات السؤال السابق .

## Supplementary Problems

## مشاكل متكاملة

(٥٠) حدد أى من المعرفات التالية صحيح .

- |                      |             |
|----------------------|-------------|
| (e) name and address | (a) record1 |
| (f) employee_number  | (b) file2   |
| (g) 123-45-6789      | (c) file    |
|                      | (d) name    |

(٥١) حدد أى من الأعداد التالية صحيح . وإذا ما كان العدد صحيحاً ؛ حدد إذا ما كان حقيقياً ، أم صحيحاً .

- |               |                |            |
|---------------|----------------|------------|
| (g) 12E12     | (d) -4.083e-67 | (a) 0.5    |
| (h) 131072    | (e) 1.         | (b) 27,822 |
| (i) 1.31072e5 | (f) 40-55      | (c) +93e12 |

(٥٢) حدد : أى من السلاسل التالية صحيح .

- |                           |                           |
|---------------------------|---------------------------|
| (d) 'Chapter 3 (Cont''d)' | (a) '8:15 P.M.'           |
| (e) '1.30172e5'           | (b) "red, white and blue" |
| (f) 'NEW YORK, NY 10020'  | (c) 'Name:                |

(٥٣) حدد : كيف يمكن للمعرفات التالية أن تصاحبها القيم الثابتة المناظرة لكل منها .

Identifier	Constant Value
month	july
fica	123-45-6789
price	\$95.00
gross	2500.00
partno	48837
bound	0.00391

(٥٤) اكتب التوضيح الذى يصاحب كل من المعرفات التالية ، والبيانات المصاحبة لكل منها .

Identifier	Data Type
period	char
terminal	boolean
status	char
index	integer
row	integer
clearance	real

(٥٥) حدد نوع كل من التعبيرات التالية . عرف أى تعبير مكتوب بطريقة خاطئة .

- |                             |                                 |
|-----------------------------|---------------------------------|
| (a) counter := 87           | (e) 2*-x+y                      |
| (b) value                   | (f) factor1*(sum1+sum2)/factor2 |
| (c) sqr(first+second+third) | (g) color = 'blue'              |
| (d) cost <= maximum         |                                 |

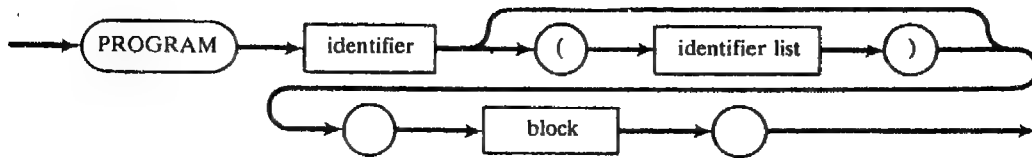
(٥٦) حدد أى من العبارات التالية بسيط ، وأيها مركب . عرف نوع كل عبارة .

- (a) net := gross-(fedtax+statetax+citytax);
- (b) BEGIN  
    tax := fedtax+statetax+citytax;  
    net := gross-tax  
END
- (c) FOR counter := start TO finish DO write(counter);
- (d) IF counter < finish THEN counter := counter+1;
- (e) root := sqrt(a+b+c+d);
- (f) write('root=',root);
- (g) GOTO 17;
- (h) new := new+old;

(٥٧) أى عبارة من عبارات السؤال السابق تتصلل بإجراءات ؟ وأيها تشير إلى نوال ؟

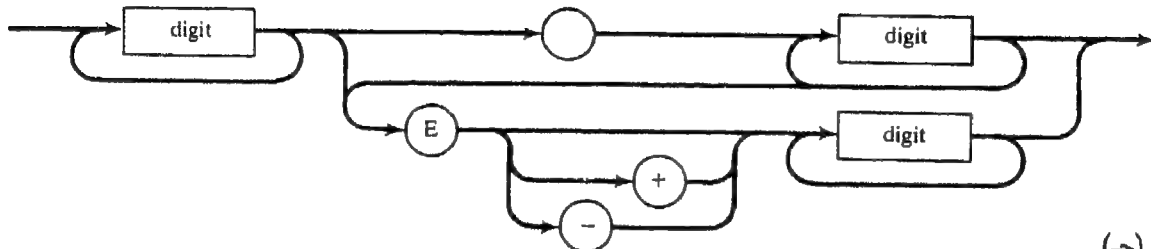
(٥٨) وضح معنى كل رسم من الرسومات التكوينية للبسكال التالية :

(أ)



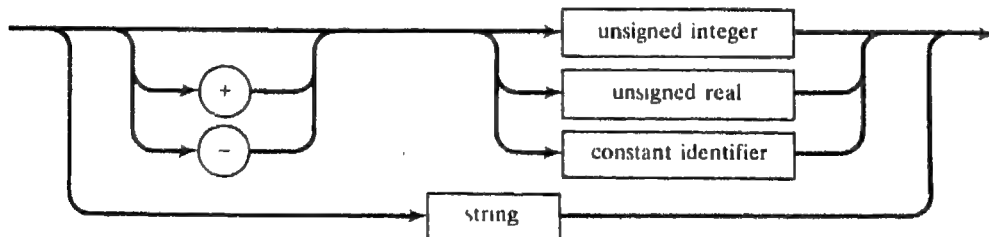
الشكل رقم (٢-٢)

(ب)



الشكل رقم (٣-٢)

(ج)



الشكل رقم (٤-٢)



## الفصل الثالث

# بيانات من النوع البسيط Simple-Type Data

لقد أصبح القارئ معتادا الآن على بعض المفاهيم الأساسية البسكال ، مما يجعلنا قادرين على الخوض في تفاصيل أكبر لبيانات بسيطة النوع . وسوف نأخذ في الاعتبار بصفة خاصة استخدامات البيانات البسيطة القياسية ، والتي تعتبر جزءا من لغة البسكال . وهذا النوع من البيانات يشمل : البيانات الصحيحة ، والحقيقية ، والحرفية ، وبيانات بوليان . كما سنناقش بالتفصيل العمليات والدوال القياسية التي يمكن استخدامها مع كل نوع من هذه الأنواع القياسية للبيانات ، كما يحتوى الفصل على معلومات إضافية عن تعبيرات واستخدامها في عبارات التحديد .

وعادة ما يشار إلى ثلاثة أنواع من البيانات القياسية ، وهي : البيانات الصحيحة ، والحرفية ، وبيانات بوليان بأنها بيانات ترتيبية ordinal ، حيث إن عناصر البيانات التي يتكون منها كل نوع من هذه الأنواع للبيانات تكون أعضاء في فئة محددة ومرتبطة . والبيانات البسيطة التي يقوم بتعريفها المستفيد ( بيانات متعددة enumerated وبيانات المدى الجزئي subrange ) تقع في هذه المجموعة أيضا . لاحظ - على أية حال - أن البيانات من النوع الحقيقي لا يمكن تقسيمها بهذه الطريقة . وعلى هذا ... فلاتقع في هذه المجموعة .

## ١ - بيانات من النوع الصحيح : 1. INTEGER-TYPE DATA

البيانات من النوع الصحيح عبارة عن كميات من أرقام صحيحة . وتقع في هذه المجموعة ثوابت ومتغيرات ودوال وتعابير من النوع الصحيح . وسبق أن ناقشنا قواعد كتابة الأرقام الصحيحة ، وقواعد تعريف وتوضيح الثوابت والمتغيرات الصحيحة . دعنا نركز الآن على المؤثرات التي يمكن استخدامها مع البيانات الصحيحة ، ثم نأخذ في الاعتبار بعد ذلك قواعد كتابة التعابير الصحيحة .

وجميع المؤثرات المستخدمة لإجراء عمليات من النوع العددي تسمى مؤثرات حسابية arithmetic operators . وهناك ستة مؤثرات حسابية يمكن استخدامها مع عناصر صحيحة يمكن التأثير عليها ، وخمسة من هذه المؤثرات تنتج عنها نتائج صحيحة ( أى نتيجة resultant من النوع الصحيح ) . أما المؤثر السادس ، فينتج عنه كمية من النوع الحقيقي . وهذه المؤثرات ملخصة على النحو التالي :

المؤثر الحسابي	الغرض منه	نوع العناصر التي يمكن أن يؤثر عليها	نوع النتيجة
+	جمع	صحيح	صحيح
-	طرح	صحيح	صحيح
*	ضرب	صحيح	صحيح
/	قسمة	صحيح	حقيقي
DIV	قسمة مع حذف الكسر العشري	صحيح	صحيح
MOD	قسمة ينتج عنها الرقم الصحيح الممثل للكسر العشري المتبقى من عملية القسمة	صحيح	صحيح

لاحظ أن مؤثر القسمة (/) هو الذى تنتج عنه كمية حقيقية ، بالرغم من أن العناصر التى يؤثر عليها صحيحة النوع . ولاحظ أيضا أنه لا يوجد مؤثر أسى ... فلا يستخدم مؤثر أسى فى البسكال .

### مثال (١-٣)

افرض أن  $a$  و  $b$  متغيران صحيحان ، حددت لهما القيمتان 13 و 5 على التوالى . وفيما يلى عدة تعبيرات صحيحة والناتج منها .

Expression	Value
$a+b$	18
$a-b$	8
$a*b$	65
$a \text{ DIV } b$	2
$a \text{ MOD } b$	3
$a/b$	2.6

وهناك قواعد معينة يجب مراعاتها عند استخدام هذه المؤثرات فى تكوين تعبيرات عددية . وبعض هذه القواعد الأكثر شيوعا - والتى تتعلق بأخر عمليتين - قد أعطيا فى المثال ( وهما  $a \text{ MOD } b$  و  $a/b$  ) . وسوف تناقش قواعد أخرى فيما بعد فى هذا الفصل .

- (١) تكون المحصلة موجبة إذا كان لكل من العنصرين نفس الإشارة ، وإلا فستكون النتيجة سالبة .
- (٢) مؤثر القسمة (/ و DIV ) ، وكذلك المؤثر لباقى خارج القسمة MOD تتطلب ألا يكون العنصر الثانى صفرا .
- (٣) استخدام المؤثر DIV مع عنصر سالب ينتج عنه حذف للكسر العشرى كلية ، أى أن النتيجة تصبح أقل من قيمتها الحقيقية .
- (٤) طبقا للصيغة القياسية من ISO ، لا يمكن أن يكون العنصر الثانى سالبا عند استخدام المؤثر MOD . وعلى أية حال تسمح عدة مترجمات بسكال بأن يكون مثل هذا العنصر سالبا . وفى مثل هذه الحالات تتحدد إشارة النتيجة ،

بحيث يتحقق الشرط التالى دائما ، بغض النظر عن إشارات العناصر الفردية .

### مثال (٢-٣)

افرض أن  $i$  و  $j$  متغيران صحيحان ، وحددت لهما القيمتان 11 و -3 على التوالى . وفيما يلى عدة تعبيرات صحيحة والقيم المناظرة لها .

Expression	Value
$i+j$	8
$i-j$	14
$i*j$	-33
$i \text{ DIV } j$	-3
$i \text{ MOD } j$	2 (nonstandard)
$i/j$	-3.6666667

فإذا ما كانت قيمة  $i$  هي -11 وقيمة  $j$  هي 3 ، فإننا نحصل على ما يلى :

Expression	Value
i DIV j	-3
i MOD j	-2
i/j	-3.6666667

وأخيرا ... إذا كانت قيمة i هي -11 وقيمة j هي -3 ، فإننا نحصل على مايلي :

Expression	Value
i DIV j	3
i MOD j	-2 (nonstandard)
i/j	3.6666667

لاحظ أن الشرط ( غير القياسي ) التالي :

$$i = (i \text{ DIV } j) * j + (i \text{ MOD } j)$$

قد تحقق في كل من المواقف سابقة الذكر .

## 2. REAL-TYPE DATA

### ٢ - بيانات من النوع الحقيقي :

تشير البيانات من النوع الحقيقي إلى عناصر بيانات تمثل كميات عددية حقيقية . وتشمل هذه الثوابت والمتغيرات والدوال والتعبيرات الحقيقية .

وهناك أربعة مؤثرات حسابية يمكن استخدامها مع عناصر من النوع الحقيقي ، وهي :

المؤثر الحسابي	الفرض منه	نوع العناصر	نوع النتيجة
+	جمع	حقيقي	حقيقي
-	طرح	حقيقي	حقيقي
*	ضرب	حقيقي	حقيقي
/	قسمة	حقيقي	حقيقي

لاحظ أنه يمكن استخدام هذه المؤثرات مع عناصر من النوع الصحيح أيضا ، إلا أن المؤثرين DIV و MOD المستخدمين مع العناصر الصحيحة لا يمكن استخدامها مع البيانات الحقيقية . ونذكر هنا مرة أخرى أنه لا يوجد مؤثر أسى في لغة البسكال .

مثال ( ٣ - ٣ )

افرض أن v1 و v2 متغيران حقيقيان ، وأنه محدد لهما القيمتان 12.5 و 0.5 على التوالي . فيما يلي بعض التعبيرات الحقيقية البسيطة والنتيجة لكل منها .

Expression	Value
v1+v2	13.0
v1-v2	12.0
v1*v2	6.25
v1/v2	25.0

وهناك قواعد معينة يجب أن تتبع عند تكوين التعبيرات الحسابية بالبيانات الحقيقية . وفيما يلي ملخصا لثلاثة قواعد شائعة الاستخدام فى تكوين تعبيرات تحتوى على عنصرين اثنين فقط . وسوف نقدم قواعد أخرى فيما بعد فى هذا الفصل .

(١) النتيجة تكون موجبة إذا كان لكل من العنصرين نفس الإشارة . وإلا فإن إشارة النتيجة تكون سالبة .

(٢) يتطلب مؤثر القسمة (/) ألا يكون العنصر الثانى صفرا .

(٣) إذا كان أحد العناصر صحيحا ، والآخر حقيقيا ؛ فإن النتيجة تكون حقيقية دائما .

مثال (٣-٤)

إذا كان  $r1$  و  $r2$  متغيرين حقيقيين لهما القيمتان -0.66 و 4.50 على التوالى . ففيما يلي عدة تعبيرات حقيقية والنتائج المناظرة لها .

Expression	Value
$r1+r2$	3.84
$r1-r2$	-5.16
$r1*r2$	-2.97
$r1/r2$	-0.1466667

مثال (٣-٥)

افرض أن  $i$  متغير من النوع الصحيح ، ومحدد له القيمة -2 ، وأن  $r$  متغير من النوع الحقيقى ، ومحدد له القيمة 1.2

التعبير  $3+i*r$

يمثل قيمة حقيقية تساوى -7.2 ، حيث  $3x (-2) \times 1.2 = 7.2$

### ٣ - بيانات من النوع الحرفى : 3. CHAR-TYPE DATA

البيانات من النوع الحرفى عبارة عن سلاسل مكونة من حرف واحد ، أو رقم واحد ، أو رمز خاص واحد ، على أن يكون موضوعا بين علامتى تنصيص . ويحتوى هذا النوع من البيانات على ثوابت ومعرفات تمثل ثوابت مكونة من حرف واحد ، أو رقم واحد ، أو رمز خاص واحد ، ومتغيرات من النوع الحرفى وبعض الدوال من النوع الحرفى .

ومجموعة الرموز التى يمكن استخدامها مع البيانات الحرفية تتغير من مترجم لمترجم آخر . وبصفة عامة ... فإن الحروف من A إلى Z ، سواء الكبيرة أم الصغيرة ، والأرقام من 0 إلى 9 ، والرموز الخاصة الشائعة الاستخدام كلها مقبولة .

مثال (٣-٦)

فيما يلي عناصر بيانات حرفية النوع صحيحة

'p' '5' 't' 'e' ' ' '!!!'

لاحظ أن آخر عنصر يمثل علامة تنصيص واحدة كما سبقت الإشارة لذلك .

ومعظم أجهزة الكمبيوتر ، وكل أجهزة الميكروكمبيوتر تستخدم مجموعة رموز الشفرة الأمريكية القياسية لتبادل المعلومات (ASCII) American Standard Codes for Information Exchange ، والتي يحدث فيها تمثيل عددي من 7 بت لكل رمز (وعلى هذا يكون هناك عدد يساوي  $2^7 = 128$  من الرموز المختلفة) . وتكون الرموز مرتبة طبقا للشفرة الخاصة بكل منها ، والأرقام بصفة خاصة تكون مرتبة طبقا لتسلسلها العددي المناسب ، أي من 0 إلى 9 ، كما أن الحروف الأبجدية تكون مرتبة طبقا لترتيبها الأبجدي ، أي من A إلى Z ، وهذا يسمح بمقارنة عناصر البيانات الحرفية التي تمثل رموز الشفرة الأمريكية القياسية لتبادل المعلومات مع بعضها طبقا لترتيبها النسبي المحدد لها في هذه المجموعة .

ويحتوي جدول ٣ - ١ على جزء من مجموعة رموز الشفرة الأمريكية القياسية لتبادل المعلومات ، موضحا المكافئ العشري لـ 7 بت الذي يمثل كل رمز من رموز المجموعة . لاحظ أن الأرقام تسبق الحروف ، وأن الحروف الكبيرة تسبق الحروف الصغيرة . لاحظ أيضا أنه هناك مجموعات صغيرة من الرموز الخاصة ، تفصل الأرقام والحروف الكبيرة والحروف الصغيرة ، أي تقع بين كل مجموعة من هذه المجموعات .

جدول (٣-١) جزء من مجموعة رموز ASCII

ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
032	blank	063	?	093	]
033	!	064	@	094	^
034	"	065	A	095	_
035	#	066	B	096	
036	\$	067	C	097	a
037	%	068	D	098	b
038	&	069	E	099	c
039	'	070	F	100	d
040	(	071	G	101	e
041	)	072	H	102	f
042	*	073	I	103	g
043	+	074	J	104	h
044	,	075	K	105	i
045	-	076	L	106	j
046	.	077	M	107	k
047	/	078	N	108	l
048	0	079	O	109	m
049	1	080	P	110	n
050	2	081	Q	111	o
051	3	082	R	112	p
052	4	083	S	113	q
053	5	084	T	114	r
054	6	085	U	115	s
055	7	086	V	116	t
056	8	087	W	117	u
057	9	088	X	118	v
058	:	089	Y	119	w
059	;	090	Z	120	x
060	<	091	[	121	y
061	=	092	\	122	z
062	>				

مثال (٣-٧)

يوضح جدول 3-1 أن الحرف A له الشفرة العشرية 65 في الشفرة الأمريكية القياسية للتبادل العشري ، وأن الحرف B له الشفرة العشرية 66 . وحيث إن 65 أقل من 66 ؛ فيعتبر A أقل من B . وعلى هذا ... فإن A يسبق B في الترتيب . وبالمثل فإن A يسبق a ، لأن 65 أقل من 97 ، كما أن 0 يسبق 1 ، لأن 48 أقل من 49 .

ويوجد استثناء هام في أجهزة الكمبيوتر IBM الكبيرة ، والتي تستخدم الشفرة الثنائية الموسعة للتبادل العشري (EBCDIC) Extended Binary Coded Decimal Interchange Code في ترتيب الرموز . وهذا هو مخطط مكون من 8 بت لكل رمز ، ويشتمل على 256 رمزا ( $2^8 = 256$ ) . ومجموعة رموز الشفرة الثنائية الموسعة للتبادل العشري هي صيغة مستقلة ، وهي ليست متوافقة مع الشفرة الأمريكية القياسية للتبادل العشري . ويوضح الجدول ٣ - ٢ جزءا من شفرة الرموز في الشفرة الثنائية الموسعة للتبادل العشري . لاحظ أن الحروف الصغيرة تسبق الحروف الكبيرة في هذه الشفرة ، كما أن الحروف الكبيرة تسبق الأرقام ، كما أنه هناك فجوات ( عدم استمرارية ) أيضا في تسلسل شفرة الأرقام . ولاتزال الحروف والأرقام مرتبة طبقا لتسلسلها الطبيعي ، بحيث يمكن على أية حال مقارنة رموز الشفرة الثنائية الموسعة للتبادل العشري طبقا لترتيبها النسبي داخل مجموعة الرموز .

جدول (٣-٢) جزء من مجموعة رموز EBCDIC

EBCDIC Value	Character	EBCDIC Value	Character	EBCDIC Value	Character
064	blank	132	d	200	H
074	]	133	e	201	I
075	.	134	f	209	J
076	<	135	g	210	K
077	(	136	h	211	L
078	+	137	i	212	M
079	!	145	j	213	N
080	&	146	k	214	O
090	[	147	l	215	P
091	\$	148	m	216	Q
092	*	149	n	217	R
093	)	150	o	226	S
094	;	151	p	227	T
095	^	152	q	228	U
096	-	153	r	229	V
097	/	162	s	230	W
108	,	163	t	231	X
109	%	164	u	232	Y
110	_	165	v	233	Z
111	>	166	w	240	0
112	?	167	x	241	1
122	:	168	y	242	2
123	*	169	z	243	3
124	@	193	A	244	4
125		194	B	245	5
126	=	195	C	246	6
127	"	196	D	247	7
129	a	197	E	248	8
130	b	198	F	249	9
131	c	199	G		

### مثال (٣-٨)

يوضح جدول ٣ - ٢ أن الشفرة العشرية للحرف A هي 193 في الشفرة الثنائية الموسعة للتبادل العشري ، وأن شفرة الحرف B هي 194 ، وحيث إن 193 أقل من 194 ، فإن A يسبق B ، وبالمثل فإن a تسبق A ، لأن 129 أقل من 193 ، كما أن 0 يسبق 1 ، لأن 240 أقل من 241

ويجب أن يحدد القارئ أى مجموعة الرموز المستخدمة في الكمبيوتر المتاح له . كما يجب أن يكون مفهوما أن الكميات الصحيحة والأرقام من النوع الحرفي تمثل مع بعضها بطريقة مختلفة داخل الكمبيوتر . ويجب أن يكون المبرمج المبتدئ حريصا في ألا يخلط بين هذه الأنواع من البيانات .

ولا يمكن استخدام أى من المؤثرات الحسابية مع البيانات الحرفية ، حيث إن عناصر البيانات الحرفية لا تمثل كميات عددية . ويمكن على أية حال مقارنة البيانات الحرفية باستخدام المؤثرات العلاقية المذكورة في القسم التالي .

### ٤ - بيانات من نوع بوليان : 4. BOOLEAN-TYPE DATA

البيانات من نوع بوليان تكون قيمتها إما صحيح أو خطأ . وتشمل هذه الفئة ثوابت ومتغيرات ودوال وتعبيرات من نوع بوليان . وتمثل القيمتان ( صحيح أو خطأ ) المستخدمتان مع بيانات بوليان فئة مرتبة تسبق خطأ فيها صحيح . ( لاحظ أن شفرة خطأ false هي 0 ، وشفرة صحيح true هي 1 ) .

وتتكون تعبيرات بوليان من مجموعة من المؤثرات من نفس نوع المؤثرات العلاقية relational operators . وتمثل هذه المؤثرات شروطا مختلفة للتساوى أو عدم التساوى . ويوجد سبعة مؤثرات علاقية في البسكال ، وسوف نأخذ في الاعتبار الستة مؤثرات التالية منها فقط في هذا الفصل :

معناه	مؤثر العلاقات
يساوى	=
لايساوى	<>
أقل من	<
أقل من أو يساوى	<=
أكبر من	>
أكبر من أو يساوى	>=

وسوف يناقش المؤثر السابع IN في الفصل الثانى عشر من الكتاب

ويمكن استخدام هذه المؤثرات الستة مع عناصر من أى نوع آخر غير نوع البوليان . وعند استخدامها مع عناصر غير عددية يشير عدم التساوى إلى ترتيب هذه العناصر طبقا للشفرة المستخدمة .

كما تستخدم بعض هذه المؤثرات ( وهي = و < و <= و > و >= ) في المقارنات ، كما هو مذكور في الفصل الثانى عشر من الكتاب .

مثال (٣-٩)

فيما يلي بعض تعبيرات بوليان بسيطة تحتوي على عناصر عددية ( الفرض أن كل من العنصرين داخل كل تعبير من نفس النوع ) .

Expression	Value
2 = 3	false
2 < 3	true
0.6 >= 1.5	false
0.6 >= -1.5	true
-4 <> 4	true
1.7 <= -2.2	false

مثال (٣-١٠)

افرض أن i و j متغيران من النوع الصحيح ، ومحدد لهما القيمتان 3 و -5 على التوالي . فيما يلي عدة تعبيرات بوليان تستخدم هذين المتغيرين .

Expression	Value
i <= 10	true
i+j > 0	false
(i-j) < (i+j)	false
i-3 = j+5	true
2*i >= i DIV 2	true
(i DIV 2) > (j+6)	false

لاحظ أن العناصر يمكن أن تكون ثوابت أو متغيرات أو تعبيرات .

مثال (٣-١١)

افرض أن ch1 و ch2 هما متغيران حرفيان ، ومحدد لهما الحرفان P و T على التوالي ، وفيما يلي عدة تعبيرات بوليان تستخدم هذين المتغيرين .

Expression	Value
ch1 = ch2	false
ch2 = 'T'	true
ch1 = 'p'	false
ch1 < ch2	true (because P precedes T)
ch2 > 'A'	true (because T succeeds A)
'W' <> ch1	true

كما تحتوي لغة البسكال على ثلاثة مؤثرات منطقية أيضا . ويسمح مؤثران منهما باستخدام عناصر من نوع بوليان في تكوين تعبيرات بوليان . أما الثالث ، فيستخدم في نفى ( أو عكس reverse ) قيمة العنصر من نوع بوليان .



والمؤثرات المنطقية هي :

المؤثر	معناه
OR	يكون التعبير صحيحا إذا كان أى من العنصرين صحيحا ، أو إذا كان كل من العنصرين صحيحا .
AND	يكون التعبير صحيحا إذا كان كل من العنصرين صحيحا فقط
NOT	يستخدم هذا المؤثر كسابقه prefix لنفى العنصر من النوع بوليان .

ولتجنب الخطأ فى ترتيب تنفيذ العمليات المنطقية ، يجب أن توضع عناصر بوليان بين قوسين .

مثال (١٢-٣)

افرض أن n متغير صحيح ، محدد له القيمة 10 ، وأن s متغير حرفي يمثل الحرف A . فيما يلي عدة تعبيرات بوليان تستخدم هذين المتغيرين .

Expression	Value
(n > 0) AND (n < 20)	true
(n > 0) AND (n < 5)	false
(n > 0) OR (n < 5)	true
(n < 0) OR (n > 20)	false
(n = 10) AND (s = 'A')	true
(n < 5) OR (s >= 'A')	true

مثال (١٣-٣)

افرض أن تعبير بوليان  $z > 6$  صحيحا ، وعلى هذا ... فإن التعبير

$\text{NOT } (z > 6)$

يكون خاطئا ، كما أن التعبير

$\text{NOT } (z <= 6)$

يكون صحيحا ، حيث إن  $(z <= 6)$  تكون خاطئة .

مثال (١٤-٣)

افرض أن ch متغير حرفي يمثل الحرف G . وعلى هذا ... فإن التعبير  $\text{ch} > 'A'$  يكون صحيحا ، كما أن التعبير  $\text{NOT } (\text{ch} > 'A')$  يكون خاطئا .

والتعبير  $\text{ch} = 'A'$  يكون خاطئا ، كما أن التعبير  $\text{NOT } (\text{ch} = 'A')$  يكون صحيحا .

وسوف نرى كيفية استخدام تعبيرات بوليان فى برامج البسكال عندما نصل إلى الفصل السادس من الكتاب .

## 5. STANDARD CONSTANTS

### ٥ - الثوابت القياسية :

يحتوى البسكال على ثلاثة معرفات قياسية تمثل ثوابت ، وهى : `maxint` و `false` و `true` . ويحدد أول هذه المعرفات `maxint` أكبر قيمة لأى كمية من النوع الصحيح . أما الثابتان الآخران ، وهما `false` و `true` ، فيمثلان قيمتين يمكن تحديدهما لعناصر بيانات من نوع بوليان . ( يجب أن يتذكر القارئ مرة أخرى أن `false` و `true` يمثلان فئة مرتبة يسبق فيها `false` ) .

## 6. STANDARD FUNCTIONS

### ٦ - الدوال القياسية :

يحتوى البسكال أيضا على عدد من الدوال القياسية التى تستخدم مع بيانات بسيطة مختلفة . ( كما يشار أيضا إلى الدوال القياسية بأنها دوال داخلية `intrinsic` ، أو دوال مبنية داخليا `built-in` ) . وتقبل بعض هذه الدوال نوعاً واحداً من المؤثرات وتعود بقيمة من نفس النوع ، بينما تقبل دوال أخرى مؤشرات من نوع واحد ، وتعود بقيمة من نوع مختلف . وفيما يلى ملخصاً لبعض هذه الدوال . ( انظر ملحق D لمزيد عن هذه الدوال ) .

الدالة	الغرض منها	نوع المؤثر (x)	نوع النتيجة
<code>abs(x)</code>	حساب القيمة المطلقة لمقدار x	صحيح أو حقيقي	مثل نوع x
<code>arctan (x)</code>	حساب الظل العكسى للزاوية x	صحيح أو حقيقي	حقيقي
<code>chr (x)</code>	تحديد الرمز الذى يعثله المتغير x	صحيح	حرفى
<code>cos(x)</code>	حساب جيب تمام الزاوية x ( x بالتقدير الدائرى )	صحيح أو حقيقي	حقيقي
<code>exp (x)</code>	حساب $e^x$ حيث $e = 2.7182818$ ، وهى أساس النظام الطبيعى للوغاريتمات	صحيح أو حقيقي	حقيقي
<code>In (x)</code>	حساب اللوغاريتم الطبيعى لـ x ( $x > 0$ )	صحيح أو حقيقي	حقيقي
<code>odd(x)</code>	تحديد ما إذا كان x زوجياً أو فردياً ( تعود بقيمة صحيح إذا كان x فردياً ، وقيمة خاطئ إذا كان x غير ذلك )	صحيح	بوليان
<code>ord (x)</code>	تحديد الرقم الصحيح ( العشري ) المقابل لشفرة الرمز x	حرفى	صحيح
<code>pred (x)</code>	تحديد ما يسبق x	حرفى أو صحيح أو بوليان	مثل نوع x
<code>round (x)</code>	تقريب قيمة x لأقرب عدد صحيح .	حقيقي	صحيح
<code>sin (x)</code>	حساب جيب الزاوية x ( x بالتقدير الدائرى )	صحيح أو حقيقي	حقيقي
<code>sqr (x)</code>	حساب مربع x	صحيح أو حقيقي	حقيقي
<code>sqrt (x)</code>	حساب الجذر التربيعى لـ x ( $x > 0$ )	صحيح أو حقيقي	حقيقي
<code>succ (x)</code>	حساب ما يلى x	حرفى أو صحيح أو بوليان	مثل نوع x
<code>trunc (x)</code>	يحذف الكسر من x	حقيقي	صحيح

وهناك دالتان قياسيتان إضافيتان ، هما `coln` و `cof` ، وتستخدمان مع ملفات البيانات . وسوف تذكر هاتان الدالتان فى الفصل الرابع من الكتاب .

والغرض من معظم الدوال التى سبق ذكرها يجب أن يكون واضحاً . وعلى أية حال ... فهناك قلة من الدوال لا يكون الغرض منها واضحاً . وفيما يلى أمثلة توضيحية لاستخدامات الدوال .

### مثال (٣-١٥)

تحتسب الدالة  $\text{abs}(x)$  القيمة المطلقة للعدد الذي يمثله المؤشر  $x$  . فإذا ما كان  $\text{diff}$  متغيراً حقيقياً ، وكان محدداً له القيمة  $-0.003$  فإن الدالة  $\text{abs}(\text{diff})$  تعود بالقيمة  $0.003$  ( لاحظ أن  $\text{diff}$  هو مؤشر فر ، الدالة ) .

### مثال (٣-١٦)

تستخدم الدالتان  $\text{chr}$  و  $\text{ord}$  لتحديد العلاقة بين أى رمز من رموز يسكال والشفرة الرقمية المناظرة له . وعلى هذا إذا كان الكمبيوتر مستخدماً لشفرة ASCII ، فإن

$\text{chr}(65)='A'$	$\text{ord}('A')=65$
$\text{chr}(112)='p'$	$\text{ord}('p')=112$
$\text{chr}(53)='5'$	$\text{ord}('5')=53$

وهكذا لاحظ أن :

$\text{ord}('A')=\text{ord}(\text{chr}(65))=65$

وكذلك

$\text{chr}(65)=\text{chr}(\text{ord}('A'))='A'$

وهكذا

### مثال (٣-١٧)

حيث إن البيانات من النوع الصحيح والحرفى والبولىان كلها تمثل فئات مرتبة ، فيمكننا أن نحدد مايسبق أو مايتبع أى عنصر بيانات داخل إحدى هذه الفئات ( أو فى أى فئة مرتبة يقوم بتعريفها المستخدم ) ، وذلك باستخدام الدالتين  $\text{succ}$  و  $\text{pred}$  .

وعلى هذا

$\text{pred}(3)=2$	$\text{succ}(3)=4$	( بيانات صحيحة )
$\text{pred}('e')='d'$	$\text{succ}('e')='f'$	( بيانات حرفية )
$\text{pred}(\text{true})=\text{false}$	$\text{succ}(\text{false})=\text{true}$	( بيانات بولىان )

لاحظ أيضاً أن فئة رموز ASCII تحقق شروطاً قبل

$\text{pred}('e')=\text{chr}(\text{ord}('e')-1)$

وكذلك

$\text{succ}('e')=\text{chr}(\text{ord}('e')+1)$

ويجب أن يفهم أن الدالتين  $\text{pred}$  و  $\text{succ}$  لا تستخدمان مع بيانات من النوع الحقيقى .

### مثال (٣-١٨)

يمكن للدالتين round و trunc أن تقبلا أعدادا حقيقية سالبة أو موجبة . وتعامل الأعداد السالبة مثل الأعداد الموجبة ، مع إضافة إشارة السالب بعد التقريب أو حذف الكسر . وعلى ذلك ، فإن :

```
round(2.3)=2      trunc(2.3)=2
round(3.7)=4      trunc(3.7)=3
round(-1.8)=-2    trunc(-1.8)=-1
round(-6.1)=-6    trunc(-6.1)=-6
```

ويمكن استخدام العديد من هذه الدوال مع بيانات أخرى غير قياسية ، أى يتم تعريفها من قبل المستخدم أيضا . وسوف نذكر هذه التطبيقات فيما بعد فى هذا الكتاب كلما كانت هناك حاجة لذلك .

كما يجب أن يتذكر القارئ مرة أخرى أن المؤشرات الموجودة فى دليل الدالة يمكن أن تكون ثوابت أو متغيرات أو تعبيرات ، أو حتى مؤشرات لدوال أخرى . والقيد الوحيد هو أن هذه المؤشرات يجب أن تكون من النوع المناسب . وسوف نتعرض للمزيد عن المؤشرات فى الفصل السابع من الكتاب ، حيث نتعرض لموضوع الإجراءات والدوال بتفصيلات أكثر .

## ٧ - المزيد عن التعبيرات : 7. MORE ABOUT EXPRESSIONS

يمكن أن يصبح التعبير معقدا فى بعض الأحيان بسبب وجود مؤثرات متعددة داخل التعبير . وفى مثل هذه الحالات يصبح من اللازم تحديد ترتيب تنفيذ العمليات المختلفة . ويمكن تحديد هذا الترتيب بواسطة أولوية المؤثر operator precedence الطبيعى ، والموجود فى لغة البسكال . ومجموعات الأولويات هى كما يلى :

الأولوية	المؤثر
1 (الأعلى)	NOT
2	*/DIV MOD AND
3	+ - OR
4 (الادنى)	* < > < <= > >= IN

وفى نفس مجموعة الأولويات تنفذ العمليات طبقا لظهورها ، أى الأولى فالتالية لها ، فالتالية لها ، من اليسار إلى اليمين .

### مثال (٣-١٩)

التعبير العددي التالى :

a-b/c\*sqrt(d)

يُناظر المعادلة الجبرية التالية

$$a - [(b/c) \times \sqrt{d}]$$

فإذا كانت قيم المتغيرات  $a, b, c, d$  لها القيم 1, 2, 3, 4 على التوالي . فسوف يمثل التعبير القيمة - 0.3333333333 ، حيث :

$$1 - [(2/3) \times \sqrt{4}] = 1 - (4/3) = -1/3$$

لاحظ أن عملية القسمة أجريت أولاً ، حيث إنها تقع في مجموعة لها أولوية مرتفعة عن أولوية الطرح . ونتيجة القسمة ضربت بعد ذلك في 4 ( طبقاً لقاعدة من اليسار إلى اليمين في أولويات التنفيذ ) . وأخيراً طرح حاصل الضرب هذا من قيمة أول متغير .

ويمكن تغيير الأولويات الطبيعية ، وذلك باستخدام الأقواس ، وذلك لأداء العمليات التي يحتويها التعبير بأي أولويات مطلوبة . وفي مثل هذه الحالات تنفذ العمليات الموجودة بين القوسين الداخليين جداً ، تليها العمليات الموجودة بين القوسين التاليين ، وهكذا .

مثال (٣-٢٠)

التعبير الحسابي التالي :

$$(a-b)/(c*\text{sqrt}(d))$$

يُناظر المعادلة الجبرية التالية :

$$(a - b)/(c \times \sqrt{d})$$

فإذا كانت قيم المتغيرات  $a, b, c, d$  هي 1, 2, 3, 4 على التوالي ، فإن التعبير الحسابي يمثل القيمة -0.166666667 ، حيث إن :

$$(1 - 2)/(3 \times \sqrt{4}) = -1/6 = -0.1666667$$

( قارن هذه النتيجة بنتيجة مثال ١٩ - ٣ ) .

مثال (٣-٢١)

اعتبر تعبير بولياني التالي :

$$(x > 0) \text{ OR } (y < 10)$$

حيث إن  $x$  و  $y$  متغيران صحيحان . ويكون هذا التعبير صحيحاً إذا ماكانت قيمة  $x$  أكبر من الصفر أو إذا كانت قيمة  $y$  أقل من 10 ( أو إذا تحقق الشرطان معاً ) . أما إذا لم يتحقق الشرطان ، فإن قيمة التعبير تكون خاطئة .

لاحظ أن الأقواس مطلوبة في هذا التعبير . فبدون الأقواس يحاول البسكال أن يقوم بالتعبير التالي :

$$0 \text{ OR } y$$

والذي لا معنى له ( نظراً لأنه لا يمكن استخدام المؤثر المنطقي OR مع عناصر عديدة ) .

### مثال (٣-٢٢)

اعتبر التعبير العددي التالي :

$$2*((a \text{ MOD } 5)*(4+(b-3)/\text{sqrt}(c+2)))$$

إذا كانت قيم المتغيرات a و b و c هي 8 و 15 و -4 على التوالي ، فسوف يتم تقويم هذا التعبير على النحو التالي :

$$2*((3)*(4+12/\text{sqrt}(-2))) = 2*(3*(4+12/4)) = 2*(3*(4+3)) = 42$$

وفي بعض الأحيان يكون استخدام الأقواس لتوضيح التعبير مفيداً ، وذلك بالرغم من أنه قد لا يكون مطلوباً وجود الأقواس . ومن ناحية أخرى ... يجب تجنب استخدام التعبيرات المعقدة جداً كلما كان ذلك ممكناً ، حيث إن مثل هذه التعبيرات تمثل مصدراً متكرراً للخطأ ( يحتوى آخر مثال على مثل هذه التعبيرات ) .

ويجب أن تتبع هذه القواعد ( بالإضافة إلى القواعد التي سبق ذكرها في هذا الفصل ) عند كتابة تعبيرات عديدة وتعبيرات بوليان

(١) لا يمكن أن تظهر معرفات غير معرفة داخل التعبير . ( وفي كلمات أخرى ... يجب أن تحدد قيمة معينة لكل معرف قبل أن يمكن استخدامه داخل التعبير ) .

(٢) ظهور إشارة سالبة تسبق أحد المعارف يكافئ ضرب المعارف في -1 ، وعلى هذا ... فإن  $a*b$  - يكافئ  $-1*a*b$  .

(٣) ممنوع ظهور المؤثرات الحسابية وراء بعضها مباشرة ، وعلى هذا ... فإن التعبير  $a*-b$  غير مسموح به . أما التعبير  $a*(-b)$  فمسموح به .

(٤) لا يمكن أن تكون العمليات الحسابية مشمولة . وعلى هذا ... فالتعبير  $2(x+y)$  خاطئ ، أما التعبير  $2*(x+y)$  ، فهو صحيح .

(٥) لا يمكن أداء العمليات الحسابية على بيانات حرفية أو بيانات بوليان . وعلى هذا ... فإن تعبيرات مثل :

$$'A' + 'B'$$

أو مثل :

$$(n > 0) + (n < 20)$$

غير مسموح بها .

(٦) يجب أن يكون هناك اتزان في الأقواس ، بمعنى أن عدد الأقواس المفتوحة يجب أن يتساوى مع عدد الأقواس المغلقة .

## 8. THE ASSIGNMENT STATEMENT

٨ - عبارة التحديد :

سبق أن رأينا أن عبارة التحديد هي عبارة بسيطة تستخدم في تحديد عناصر بيانات لمتغيرات . وتكتب هذه العبارة على النحو التالي :

variable := data item

ويمكن أن يكون عنصر البيانات فرديا ( مثل ثابت ، أو متغير آخر ، أو دليل دالة ) أو يكون تعبيرا . ويجب على أية حال أن يكون عنصر البيانات من نفس نوع المتغير المحدد له . ( وهناك استثناء واحد لهذه القاعدة ، وهو أنه يمكن تحديد عنصر بيانات صحيح لمتغير حقيقي ) .

مثال (٣-٢٣)

فيما يلي عبارة تحديد تقليدية لمساحة الدائرة

```
area := 3.14159*sqr(radius);
```

وتسبب هذه العبارة تحديد قيمة للتعبير

```
3.14159*sqr(radius)
```

وتحدد هذه القيمة للمتغير area . وعلى هذا ... فإذا كانت قيمة نصف القطر radius هي 10.0 ، فإن القيمة التي تحدد للمتغير المساحة area هي 314.159 .

والفاصلة المنقوطة الموجودة في نهاية العبارة ليست في واقع الأمر جزءا من العبارة نفسها ، ولكنها فاصل separator يوضح أن العبارات انتهت ، وأن مايلي هذه الفاصلة هو بداية عبارة جديدة . وتستخدم الفاصلة المنقوطة مع كل عبارات البسكال ، ولا يقتصر استخدامها على أى نوع من العبارات ، مثل عبارات التحديد .

وعادة ماتتشابه عبارات التحديد العديدة مع المعادلات الجبرية . وليس هذا صحيحا دائما على أية حال كما يظهر ذلك من المثال التالي .

مثال (٣-٢٤)

اعتبر عبارة التحديد التالية :

```
count := count+1;
```

تسبب هذه العبارة في زيادة القيمة الحالية للمتغير count بمقدار 1. ( وهذا يسمى بالعداد ) . ومن ناحية الجبر ، فإن هذه العبارة لأمعنى لها .

ويمكن تحقيق نفس الشئ بكتابة مايلي :

```
count := succ(count);
```

حيث succ هي دالة قياسية ، كما سبق ذكره .

ولاحتياج عبارة التحديد أن تقتصر على البيانات العديدة فقط كما هو موضح في المثالين التاليين .

مثال (٣-٢٥)

افرض أن state هو متغير حرفي . وعلى هذا ... فإن عبارة التحديد التالية :

```
state := 'S';
```

تحدد أن قيمة state هي الحرف S .

مثال (٣-٢٦)

افترض أن flag هو متغير بولياني ، وأن x , y هما متغيران صحيحان . وعلى هذا ... فإن العبارة التالية :

$$\text{flag} := (x > 0) \text{ OR } (y < 10);$$

تتسبب في وضع قيمة صحيح أو خاطئ للمتغير flag طبقا لقيمة التعبير البولياني . وعلى هذا ... فإن قيمة flag تكون صحيح إذا كانت قيمة x أكبر من صفر ، أو إذا كانت قيمة y أقل من 10 ( أو إذا تحقق الشرطان ) وإلا فإن قيمة flag تكون خاطئ .



## Review Questions

## أسئلة للمراجعة :

- (١) ماذا تمثل البيانات من النوع الصحيح ؟
- (٢) ماهو المؤثر الحسابى ؟ أى المؤثرات الحسابية يمكن استخدامها مع البيانات الصحيحة ؟ وماهو الغرض من استخدام كل من هذه المؤثرات ؟
- (٣) ماهو الفرق بين مؤثر القسمة (/) ومؤثر الحذف (DIV) عند استخدام عناصر بيانات صحيحة ؟
- (٤) لخص قواعد استخدام أحد المؤثرات الحسابية مع عناصر بيانات صحيحة ، ولخص بصفة خاصة القيود الموجودة على استخدام مؤثرى القسمة (/ و DIV) والمؤثر MOD .
- (٥) ماذا تمثل البيانات من النوع الحقيقى ؟
- (٦) ماهى المؤثرات الحسابية التى يمكن استخدامها مع البيانات الحقيقية ؟ وماهو الغرض من استخدام كل من هذه المؤثرات ؟
- (٧) لخص قواعد استخدام أى مؤثر حسابى مع عنصرى بيانات حقيقيين .
- (٨) مانوع النتيجة التى يتم الحصول عليها اذا ما استخدم احد المؤثرات الحسابية مع عنصر بيانات صحيح وعنصر بيانات حقيقى ؟
- (٩) ماذا تمثل البيانات من النوع الحرفى ؟ وماهى الرموز التى يمكن استخدامها مع البيانات الحرفية ؟
- (١٠) ماذا تعنى مجموعة رموز ASCII ؟ ومامدى استخدام هذه المجموعة ؟
- (١١) ماهى الصورة العامة لترتيب الرموز فى مجموعة رموز ASCII ؟
- (١٢) ماهى مجموعة رموز EBCDIC ؟ وفى أى أنواع أجهزة الكمبيوتر يمكن أن توجد هذه المجموعة ؟
- (١٣) ماهى الصورة العامة لترتيب الرموز فى مجموعة رموز EBCDIC ؟
- (١٤) مانوع العمليات الذى يمكن اداؤها على البيانات الحرفية ؟ وماهى المؤثرات المستخدمة ؟
- (١٥) ماذا تمثل البيانات من نوع بوليان ؟
- (١٦) كيف يتم ترتيب عناصر البيانات من نوع بوليان ؟
- (١٧) ماهى المؤثرات العلاقية ؟ وماهو الغرض من كل منها ؟ ومع أى نوع من أنواع عناصر البيانات تستخدم هذه المؤثرات ؟ وماهى النتيجة التى يتم الحصول عليها فى كل حالة ؟
- (١٨) ماهو تفسير مؤثر علاقى من نوع عدم التساوى ( مثل < ) والمستخدم مع عناصر حرفية ؟
- (١٩) ماهى المؤثرات المنطقية ؟ وماهو الغرض من كل منها ؟ ومع أى عناصر بيانات يمكن استخدامها ؟ وماهو نوع النتيجة التى يتم الحصول عليها ؟
- (٢٠) ماهى الثلاثة ثوابت القياسية الموجودة فى البسكال ؟ ومانوع عنصر بيانات كل ثابت من هذه الثوابت ؟ وماالغرض من كل هذه الثوابت ؟

- (٢١) ماهو الدوال القياسية ؟ وماهى الأسماء الأخرى لها ؟
- (٢٢) ماهو الغرض من دالة abs ؟ ومع أى نوع من أنواع المؤشرات تستخدم هذه الدالة ؟ وماهو نوع النتيجة التى يتم الحصول عليها ؟
- (٢٣) ماهو الغرض من دالتى chr و ord ؟ ومع أى نوع من أنواع المؤشرات تستخدم هاتان الدالتان ؟ وماهو نوع النتائج التى يتم الحصول عليها ؟
- (٢٤) ماهو الغرض من دالتى pred و succ ؟ ومع أى نوع من أنواع المؤشرات تستخدم هاتان الدالتان ؟ وماهو نوع النتائج التى يتم الحصول عليها ؟
- (٢٥) عند استخدام مجموعة رموز ASCII ، فماهى العلاقة الموجودة بين دالتى chr و ord ، ودالتى pred و succ ؟ ولماذا لا تكون هذه العلاقة صحيحة مع كل مجموعات الرموز ؟
- (٢٦) ماهو الغرض من استخدام دالتى round و trunc ؟ ومع أى نوع من أنواع المؤشرات تستخدم هاتان الدالتان ؟ وماهو نوع النتائج التى يتم الحصول عليها ؟
- (٢٧) كيف تعامل دالتا round و trunc المؤشرات السالبة ؟
- (٢٨) وضح ماهو معنى أولوية المؤثر operator precedence ، ولخص تكوينه .
- (٢٩) ماهو ترتيب تنفيذ العمليات الموجودة داخل إحدى مجموعات الأولويات ؟
- (٣٠) كيف يمكن تغيير الأولويات المعتادة للتنفيذ ؟
- (٣١) ماهى أولويات تنفيذ العمليات الموجودة داخل تعبير يحتوى على عدة أقواس متداخلة ؟
- (٣٢) تحت أى ظروف يجب أن توجد الأقواس داخل التعبير ؟ وتحت أى ظروف يجب تجنب وجود الأقواس داخل التعبير ؟
- (٣٣) لخص القواعد التى تجب مراعاتها عند عمل تعبيرات عددية وتعبيرات بوليان .
- (٣٤) ماهو الغرض من عبارة التحديد ؟ وماهى القيود الموجودة على نوع البيانات التى تحدد للمتغير ؟
- (٣٥) ماهو الغرض من الفاصلة المنقوطة التى تظهر فى نهاية عبارة التحديد ؟ وهل الفاصلة المنقوطة جزء فعلا من عبارة التحديد ؟
- (٣٦) هل هناك علاقة بين عبارة تحديد للنوع الصحيح والمعادلة الجبرية ؟

## Solved Problems

مسائل محلولة :

(٢٧) افرض أن  $a, b, c$  متغيرات عددية محدد لها القيم التالية :

المتغير	النوع	القيمة
a	حقيقي	5.7
b	حقيقي	8.2
c	صحيح	7
d	صحيح	4

وتستخدم هذه المتغيرات في التعبيرات العددية التالية . حدد في كل حالة من هذه الحالات نوع التعبير ، والقيمة التي يمثلها .

التعبير العددي	النوع	القيمة
$\text{sqr}(a+b)/(c+d)$	حقيقي	17.5645
$6*(c \text{ MOD } d)$	صحيح	18
$6.0*(c \text{ MOD } d)$	حقيقي	18.0
$(c \text{ DIV } d)+(c \text{ MOD } d)$	صحيح	4
$(c \text{ MOD } d)/2$	حقيقي	1.5
$\text{trunc}(a-b)$	صحيح	-2
$\text{trunc}(a-b) \text{ DIV } c$	صحيح	0
$(c \text{ DIV } d)/(-a)$	حقيقي	-0.17543860

(٢٨) فيما يلي العديد من تعبيرات بوليان . حدد القيمة التي يمثلها كل منها . افرض أن المتغيرات  $a, b, c, d$  لها نفس قيم السؤال السابق ، وأن الحرف عبارة عن متغير حرفي محدد له الرمز  $w$  .

تعبير بوليان	القيمة
$a < b$	صحيح
$\text{letter} = '\$'$	خطأ
$'q' < 'r'$	صحيح
$\text{abs}(a-b) > 2.0$	صحيح
$(c \leq d) \text{ OR } (\text{letter} = 'Z')$	خطأ
$\text{NOT } (5 = \text{pred}(6))$	خطأ
$100*(c+d) = \text{maxint}$	خطأ
$\text{true}$	صحيح
$(c > 0) \text{ AND } (c \leq 7)$	خطأ

(٣٩) فيما يلي عدة تعبيرات عددية وتعابير بوليان . بعضهما خاطئ . حدد هذه التعابير الخاطئة ، وسبب الخطأ .

التعبير	الصحة
$2 * a * b / (c - 1) - u / \text{abs}(3 * (v - w))$	- صحيح .
$2 * a * b / (c - 1) - u / \text{abs}(3 * (v - w))$	- غير صحيح ( مؤثرين متتاليين )
'a' + 'b' + 'c'	- غير صحيح ( لا يمكن استخدام مؤثرات حسابية مع أنواع بيانات حرفية )
$\text{succ}('E') = 'F'$	- صحيح .
$x > 0 \text{ AND } x < 100$	- غير صحيح ( يجب أن توضع مؤثرات بوليان بين أقواس )
$(a + b + c) <= (c + d + e)$	- صحيح .
$2 * n + 1 <> 'a'$	- غير صحيح ( العوامل غير متوافقة )
$\text{round}(99.7) = 100$	- صحيح .
$((x - y) / 3 + (\text{abs}(u - v)))$	- غير صحيح ( الأقواس غير متوازنة )
$(-5(x - y) / 3 + \text{abs}(u - v))$	- غير صحيح ( علامة الضرب غير موجودة )

## Supplementary Problems

## مشاكل متكاملة :

(٤٠) افرض أن a , b , c , d هي متغيرات عددية محدد لها القيم التالية :

Variable	Type	Value
a	integer	8
b	integer	5
c	real	4.3
d	real	0.8
e	real	-2.2

تستخدم هذه المتغيرات في التعابير العددية التالية . حدد نوع كل تعبير من هذه التعابير ، والقيمة التي يمثلها .

(a) $(b - a) / \text{sqrt}(d - e)$	(d) $(a - 2 * b) * \text{trunc}(3 * c - d + 2 * e)$
(b) $\text{round}((c + d) / e)$	(e) $0.01 * (a - b)$
(c) $(a \text{ DIV } b) / (a \text{ MOD } b)$	(f) $\text{trunc}(3 * \text{sqrt}(\text{abs}(d + e)))$

(٤١) فيما يلي عدة تعابير بوليان . حدد القيمة التي يمثلها كل من هذه التعابير . افرض أن المتغيرات

a , b , c , d , e لها نفس القيم الموجودة في المسألة السابقة ، وأن n , y متغيرات حرفية محدد لها الحرفان N , Y على التوالي .

(a) $c < d + e$	(e) true OR false
(b) $(y = 'Y') \text{ AND } (n = 'N')$	(f) NOT $(y < 'z')$
(c) $\text{trunc}(c + d) <= 10.0$	(g) $(a >= 100) \text{ AND } (b <= \text{maxint})$
(d) $(a = 8) \text{ OR } (b = 8)$	(h) odd(a - b)

(٤٢) حدد قيمة كل تعبير من التعبيرات التالية افرض أن الرموز مأخوذة طبقاً لنظام شفرة ASCII .

- |                              |                              |                                  |
|------------------------------|------------------------------|----------------------------------|
| (a) <code>abs(-4,667)</code> | (i) <code>succ(false)</code> | (p) <code>trunc(2.2)</code>      |
| (b) <code>chr(67)</code>     | (j) <code>odd(10)</code>     | (q) <code>trunc(2.8)</code>      |
| (c) <code>ord('e')</code>    | (k) <code>odd(15)</code>     | (r) <code>trunc(-2.2)</code>     |
| (d) <code>pred(10)</code>    | (l) <code>round(2.2)</code>  | (s) <code>trunc(-2.8)</code>     |
| (e) <code>pred('e')</code>   | (m) <code>round(2.8)</code>  | (t) <code>chr(ord('8')+4)</code> |
| (f) <code>pred(true)</code>  | (n) <code>round(-2.2)</code> | (u) <code>ord(pred('A'))</code>  |
| (g) <code>succ(10)</code>    | (o) <code>round(-2.8)</code> | (v) <code>chr(ord('g'))</code>   |
| (h) <code>succ(e)</code>     |                              |                                  |

(٤٣) أى إجابة من إجابات المسألة السابقة سوف تتغير إذا ما استخدمت مجموعة رموز EBCDIC ؟

(٤٤) فيما يلي عدة تعبيرات عددية وتعابير بوليان . بعض هذه التعبيرات خاطئ . حدد التعبيرات الخاطئة ، وسبب الخطأ .

- |  |   |
|--|---|
| (a) <code>sum DIV 0.005</code>                 | (h) <code>slope &lt;&gt; m*x+b</code>             |
| (b) <code>200*sum</code>                       | (i) <code>sqrt(a+b) &lt; succ('G')</code>         |
| (c) <code>'100'+'27'+440'</code>               | (j) <code>(a+abs(c1+c2))/((c1+c3)*(c2+c3))</code> |
| (d) <code>true = false</code>                  | (k) <code>2*maxint+3</code>                       |
| (e) <code>(a-b)/(2g+7)</code>                  | (l) <code>round(a*-b)</code>                      |
| (f) <code>'D'=pred('E')</code>                 | (m) <code>odd(12.82)-6.6</code>                   |
| (g) <code>value = 0.0 OR count &lt; 100</code> | (n) <code>-b+sqrt(sqr(b)-4*a*c)/(2*a)</code>      |

(٤٥) يحتوى أحد برامج البسكال المتغيرات التالية :

Variable	Data Type
gross	real
net	real
tax	real
employee	integer
status	char
sex	char
exempt	boolean

بين كيف يتم توضيح هذه المتغيرات فى البرنامج .



## الفصل الرابع

### إدخال وإخراج البيانات

## Data Input and Output

يهتم هذه الفصل بطرق قراءة البيانات داخل الكمبيوتر ، وكتابة الكمبيوتر للمخرجات . وسوف نرى أنه يمكن تحقيق ذلك بسهولة ، وذلك باستخدام عبارات خاصة للمدخلات والمخرجات موجودة في لغة البسكال . وبعد أن نتعلم كيفية أداء عمليات المدخلات والمخرجات هذه ، فإننا نكون قادرين على كتابة العديد من برامج البسكال البسيطة .

### ١ - ملفات المدخلات والمخرجات : 1. INPUT AND OUTPUT FILES

دعنا نعتبر طريقة تحويل بيانات المدخلات إلى الكمبيوتر ، وتحويل بيانات المخرجات من الكمبيوتر . وسوف نعتبر بيئة ( غير متداخلة ) لجهاز كمبيوتر كبير .

يجب وضع عنصر بيانات المدخلات في ملف منفصل ، يسمى ملف مدخلات input file قبل تشغيل البرنامج المناظر له . وفي معظم الأحوال يتم انتاج هذا الملف عن طريق منقح للنصوص ، كالمقح المذكور في الفصل الخامس من الكتاب . كما يمكن أيضا إدخاله في الكمبيوتر عن طريق بطاقات مثقبة ، وذلك باستخدام آلة التنقيب وقارئ البطاقات . وفي أى حالة من هذه الحالات تخزن عناصر البيانات على التوالي بنفس الترتيب الذي تم إدخالها به . وسوف تجمع عناصر البيانات الفردية في أسطر lines منطقية ، حيث يناظر كل من هذه الأسطر سطرا من البيانات التي تم إدخالها ، أو يناظر بطاقة مثقبة .

وتنقل بيانات المخرجات من ذاكرة الكمبيوتر إلى ملف مخرجات output file ، والذي يشبه ملف المدخلات ، لكنه عكسه وتجمع عناصر البيانات مرة أخرى على التوالي ، وب نفس الترتيب التي كتبت به . سوف تجمع عناصر البيانات مرة أخرى في أسطر منطقية ، والتي تناظر الأسطر الطبيعية ( أو الواقعية ) للمخرجات عند طباعة البيانات أو عرضها على الشاشة .

ويجب أن يكون مفهوما أن ملف بيانات المدخلات و ملف بيانات المخرجات يخزان داخل الكمبيوتر ككيانين منفصلين ، فهما ليسا أجزاء من برنامج البسكال . وتصاحب هذه الملفات برنامج البسكال على أية حال بتشغيلها كمؤشرات داخل عنوان البرنامج .

### مثال (٤-١)

يحتوى برنامج بسكال على العنوان التالى للبرنامج :

```
PROGRAM payroll(input,output);
```

ويحدد هذا العنوان أن اسم البرنامج هو payroll ، وأن البرنامج يستخدم ملف مدخلات و ملف مخرجات .

ولا يحتاج البرنامج إلى أن يستخدم كل من ملفي مدخلات ومخرجات ، وذلك بالرغم من أن معظم البرامج تفعل ذلك ، حيث إن معظم البرامج تتطلب بيانات مدخلات ، وتنتج بيانات مخرجات بعد تشغيلها .

### مثال (٤ - ٢)

يستخدم برنامج بسكال اسمه primes فى إنتاج أول أثني عشر عددا أوليا . ويحتوى هذا البرنامج على العنوان التالى :

```
PROGRAM primes(output);
```

لاحظ أن هذا البرنامج يستخدم ملف مخرجات فقط ، حيث إنه لا يحتاج إلى بيانات مدخلات . فإذا ما عدل البرنامج على أية حال ، بحيث إنه ينتج أول  $n$  من الأعداد الأولية ، حيث  $n$  إحدى القيم التى يتم إدخالها للبرنامج ، فيجب أن يتغير عنوان البرنامج ليأخذ الصورة التالية :

```
PROGRAM primes(input,output);
```

ومثل هذا التعديل يعتبر فكرة جيدة ، حيث إن البرنامج يصبح عاما ، ولا يقتصر على عدد محدد من الأعداد الأولية .

ويجب أن يتذكر القارئ أنه هناك بعض صيغ متداخلة للغة البسكال تسمح بأداء عمليات المدخلات والمخرجات مباشرة أثناء تنفيذ البرنامج . وفى مثل هذه الحالات لا تكون هناك حاجة إلى ملفات المدخلات والمخرجات . وهذا شائع الاستخدام مع أجهزة الميكروكمبيوتر بصفة خاصة .

## ٢ - عبارة اقرأ : 2. THE READ STATEMENT

تستخدم عبارة اقرأ لقراءة عناصر بيانات من ملف المدخلات ، ولتحديد هذه العناصر لمتغيرات صحيحة أو حقيقية أو حرفية . وتكتب هذه العبارة على النحو التالى :

```
read(input variables)
```

وتفصل متغيرات المدخلات بواسطة فواصل . ( لاحظ أنه لا يمكن أن تظهر متغيرات بوليان داخل قائمة متغيرات المدخلات ) .

### مثال (٤ - ٣)

فيما يلي عبارة اقرأ نمطية

```
read(a,b,c);
```

تتسبب هذه العبارة فى قراءة ثلاثة عناصر بيانات من ملف مدخلات ، وتحدد هذه العناصر لثلاثة متغيرات هى :  $a$  ,  $b$  ,  $c$  على التوالى .

وتقرأ عناصر البيانات من ملف المدخلات وتحدد للمتغيرات المناظرة لها بنفس الترتيب الذى تخزن به . ويجب أن يكون كل متغير من نفس نوع عنصر البيانات المناظر له . ( استثناء : يمكن تحديد عدد صحيح لمتغير حقيقي ) . ويمكن قراءة كل عنصر بيانات مرة واحدة فقط .

### مثال (٤ - ٤)

فيما يلي جزء من برنامج بسكال

(انظر الصفحة التالية)



```
VAR a,b : real;
    i,j : integer;
    p,q : char;
    .
    .
    .
    read(a,b,i,j,p,q);
```

تسبب عبارة اقرأ في قراءة عددين حقيقيين ، وعددين صحيحين ، ورمزين فرديين من ملف المدخلات مع تحديدها للمتغيرات a,b,i,j,p,q على التوالي .

يجب الحذر عند وضع عناصر بيانات المدخلات . فيجب فصل عناصر البيانات العددية عن بعضها بفراغات أو بمؤشر انتهاء السطر . وحتى نكون أكثر دقة ، يجب أن يسبق عنصر البيانات العددي فراغ واحد أو أكثر ، أو مؤشر واحد لانتهاء السطر أو أكثر . ويمكن كتابة الأعداد الحقيقية باستخدام أس ، أو بدون استخدامه . والأكثر من هذا ... يمكن كتابة الأعداد الحقيقية ، والتي تمثل كميات صحيحة ( مثل 1.0 ) في الصورة الصحيحة ( أى 1 ) . ويمكن أن تسبق أى عدد إشارة سالبة أو إشارة موجب ، على ألا يكون هناك فراغ بين الإشارة والعدد .

ويجب معاملة البيانات الحرفية بطريقة مختلفة بعض الشيء ، حيث إن كل الرموز معنوية . ويجب ألا يوضع عنصر البيانات الحرفي بين علامتى تنصيص بصفة خاصة . كما أنه لا يمكن فصل عنصر البيانات الحرفي عن عنصر البيانات الذى يسبقه باستخدام فراغ أو مؤشر نهاية السطر ، حيث إن هذه الفواصل تفسر ( بطريق خاطئ ) على أنها عناصر بيانات . ( فيفسر مؤشر نهاية السطر على أنه فراغ إذا ماحدد لمتغير حرفي ) .

#### مثال (٤-٥)

؟ اعتبر مرة أخرى جزء برنامج البسكال الموجود فى المثال السابق ، أى :

```
VAR a,b : real;
    i,j : integer;
    p,q : char;
    .
    .
    .
    read(a,b,i,j,p,q);
```

افرض أن القيم التالية سوف تحدد للمتغيرات

Variable	Value
a	12500
b	-14.8
i	5
j	-9
p	X
q	Y

يمكن إدخال عناصر البيانات فى ملف المدخلات على النحو التالى :

12500.0 -14.8 5 -9XY

أو يمكن إدخالها على النحو التالي :

12500 -14.8 +5 -9XY

أو على النحو التالي :

1.25e4 -1.48e1 5 -9XY

إذا ما ظهرت عبارتان read وراء بعضهما ، فإن العبارة الثانية تبدأ من حيث انتهت العبارة الأولى . وبدقة أكثر... سوف تبدأ عبارة read الثانية بقراءة عنصر البيانات الذي يتبع آخر عنصر بيانات تم قراءته بعبارة read السابقة . وعلى هذا ... فإن عبارة read الجديدة لا يلزم لها أن تبدأ بقراءة سطر جديد من أسطر البيانات .

#### مثال (٤-٦)

دعنا نعتبر برنامج البسكال المذكور في كل من المثالين السابقين . افترض أن عبارة read الموجودة في هذا البرنامج استبدلت بالعبارتين التاليتين :

```
read(a,b,i);
read(j,p,q);
```

لا تكون هناك حاجة إلى تغيير ملف المدخلات ، حيث إن أول عبارة read تقرأ أول ثلاث قيم موجودة في السطر ، وثاني عبارة read تقرأ آخر ثلاث قيم موجودة في السطر .

### ٣ - عبارة READLN : 3. THE READLN STATEMENT

تستخدم عبارة readln مثل عبارة read في قراءة عناصر بيانات من ملف مدخلات . وتحدد هذه العناصر لتفسيرات صحيحة أو حقيقية أو حرفية . وتكتب هذه العبارة على النحو التالي :

```
readln(input variables)
```

والفرق بين العبارتين يقع في أن عبارة readln تتسبب في أن تبدأ read التالية ( وليست read الحالية ) في بدء القراءة من سطر بيانات جديد ، بينما تسمح عبارة read لعبارة read أو readln التالية بأن تبدأ بالقراءة من نفس السطر .

#### مثال (٤-٧)

فيما يلي جزء من برنامج بسكال

```
VAR p1,p2,p3,p4 : integer;
.
.
.
read(p1,p2);
read(p3,p4);
```

افترض أن ملف المدخلات يحتوي على الأعداد الثمانية التالية ، مرتبة في سطرين :

1	2	3	4
5	6	7	8

تتسبب عبارات read فى تحديد الأرقام 1 , 2 , 3 , 4 للمتغيرات p1 , p2 , p3 , p4 على التوالى .

أما إذا استبدلت عبارات read بعبارات readln ، أى يأخذ جزء البرنامج الشكل التالى :

```
VAR p1,p2,p3,p4 : integer;
.
.
.
readln(p1,p2);
readln(p3,p4);
```

فسوف يتحدد 1 , 2 للمتغيرين p1 , p2 ، كما فى الحالة السابقة . أما p3 , p4 ، فسوف يتحدد لهما 5 , 6 على التوالى ( وذلك من بيانات مدخلات السطر الثانى ) .

لاحظ أن أول عبارة readln هى التى تسببت فى تحديد قيم p3 , p4 من السطر الثانى من أسطر المدخلات .

إذا ماسبقت عبارة readln عبارة read ، فسوف تبدأ عبارة readln عند النقطة التى تنتهى عندها عبارة read . وعلى هذا ... لايلزم لعبارة readln أن تبدأ بقراءة سطر بيانات جديد . ( لاحظ أن عبارة readln تتصرف فى هذا الموقف مثل عبارة read تماما ) .

#### مثال ( ٤ - ٨ )

افرض أن عبارات read فى المثال رقم ٤ - ٧ كتبت على النحو التالى :

```
read(p1,p2);
readln(p3,p4);
```

فى هذه الحالة تتحدد القيم 1 , 2 , 3 , 4 للمتغيرات p1 , p2 , p3 , p4 على التوالى ، حيث إن عبارة readln تبدأ بقراءة القيمة الثالثة من أول سطر بيانات . وعلى أية حال ... فإن عبارة read أو عبارة readln التالية تبدأ بقراءة 5 من السطر الثانى من أسطر البيانات .

وعبارة readln مريحة فى قراءة البيانات سطرا بسطر ، كما فى حالة تشغيل الدفوعات المعتاد .

#### مثال ( ٤ - ٩ )

يحتوى أحد برامج البسكال على عبارات readln التالية :

```
readln(a,b);
readln(c,d);
readln(e,f);
```

افرض أن جميع المتغيرات من النوع الصحيح ، وأن ملف المدخلات يحتوى على الأعداد التالية :

```
1 2 3
4 5 6
7 8 9
```

سوف تتحدد القيم 4 , 5 للمتغيرين c , d ، والقيم 7 , 8 للمتغيرين e , f .

وليس من الواضح أى قيم سوف تتحدد للمتغيرين  $a, b$  ، حيث إنه غير معروف أين تبدأ أول عبارة `readln` . فإذا مابدأت العبارة فى بداية السطر ، فسوف تتحدد القيم 1, 2 للمتغيرين  $a, b$  على التوالى . ومن ناحية أخرى ... إذا ماسبق هذه العبارة عبارة `read` ، ولتكن

`read(x);`

فسوف يتحدد 1 للمتغير  $x$  ، ويتحدد 2, 3 للمتغيرين  $a, b$  على التوالى .

#### ٤ - دالّتا EOF , EOLN : 4. THE EOLN AND EOF FUNCTIONS

يحتوى البسكال على دالتين قياسيتين ، هما `eof` , `eoln` . وهما مفيدتان فى قراءة بيانات المدخلات . وأول هاتين الدالتين `eoln` تعود بقيمة بوليان `true` إذا ما انتهى السطر الذى تقرأ منه البيانات . وإلا فإنها تعود بقيمة `false` .

وتوضع قيمة `eoln` مساوية `false` بصورة تلقائية عندما تظهر بداية سطر بيانات جديد ، بافتراض أن السطر التالى ليس خالياً ( أى أن السطر التالى يجب أن يكون محتوياً على عناصر بيانات واحد على الأقل ) . وعلى هذا ... فتسمح هذه الدالة بقراءة عدد غير محدد من عناصر البيانات من أى سطر ، وذلك بقراءة البيانات من نفس السطر ، حتى تعود `eoln` بقيمة `true` . ويمكننا عند ذلك أن نعيد هذا الاجراء بالنسبة للسطر الثانى من أسطر البيانات ، وهكذا . وسوف توضح الطريقة الدقيقة التى يحدث بها ذلك فيما بعد فى هذا الكتاب ( فى الفصل السابع ، والفصل التاسع ، والفصل العاشر ) .

وعادة ماتستخدم دالة `eof` لاكتشاف نهاية ملف المدخلات ( أى تحديد نهاية الملف ) . وتعود هذه الدالة بقيمة بوليان `true` عندما تكتشف نهاية الملف ، وإلا فإنها تعود بقيمة `false` . وسوف يوضح استخدام دالة `eof` فى الفصل الحادى عشر من الكتاب .

وعادة ماتستخدم دالة `eof` مع دالة `eoln` لقراءة عدد غير محدد من عناصر البيانات من ملف المدخلات . ويمكن اكتشاف انتهاء كل سطر بواسطة دالة `eoln` ، كما يمكن اكتشاف انتهاء الملف ( أى انتهاء آخر سطر ) باستخدام دالة `eof` . وسوف نرى كيف يمكن تحقيق ذلك فى الفصل الحادى عشر .

#### ٥ - عبارة اكتب : 5. THE WRITE STATEMENT

تستخدم عبارة ( اكتب ) لكتابة عناصر بيانات فى ملف المخرجات ، وتكتب هذه العبارة على النحو التالى :

`write(output data items)`

ويمكن لعناصر بيانات المخرجات أن تكون سلاسل أو ثوابت عديدة ، أو قيما لمتغيرات أو تعبيرات . ويمكن أن تكون من النوع الصحيح أو الحقيقى أو الحرفى أو البوليانى . ( ويجب أن توضع كل سلسلة بين علامتى تنصيص ، كما سبق ذكره فى الفصل الثانى من الكتاب ) . ويجب أن تفصل عناصر البيانات بواسطة فواصل ، إذا ماكان هناك أكثر من عنصر بيانات واحد .

مثال (٤-١٠)

فيما يلى مثالاً نمطياً لعبارة اكتب

`write('x=',x);`

تتسبب هذه العبارة فى كتابة القيمة العددية للمتغير  $x$  فى ملف المخرجات ، وذلك مع الاسم المناظر لها . فإذا ماكانت  $x$  تمثل 123.456 ، فسوف يكتب مايلى فى ملف المخرجات .

`x=1.2345600E+02`

وبالمثل فإن عبارة اكتب التالية :

```
write('sum=',a+b);
```

تسبب في كتابة القيمة العددية لنتائج التعبير a+b في ملف المخرجات مع الاسم المناظر لها . وعلى هذا ... فإذا كان a , b يمثلان 3 , 1- على التوالي ، فسوف ينتج عن عبارة اكتب المخرجات التالية :

```
sum=      2
```

ويمكن عرض الأعداد الحقيقية بعدة طرق مختلفة ، بالرغم من أن الطريقة النمطية هي التمثيل العلمي ، كما هو موضح في المثال السابق . وتمثل عناصر بيانات بوليان بمعرفات قياسية true ، أو false طبقا لقيمتها .

وتستخدم معظم صيغ البسكال عرضا قياسيا للحقل field width ( أى عدد الخانات التي يشغلها الحقل ) لتمثيل عناصر البيانات الصحيحة والحقيقية والبوليان . وسوف يختلف عرض الحقل هذا لكل نوع من أنواع البيانات . والأكثر من هذا ... فإن عرض الحقل المستخدم لنوع بيانات معين يتغير من صيغة لصيغة أخرى من صيغ البسكال . وبصورة تقليدية يمكن أن يكون للبيانات الصحيحة عرض قياسي للحقل ، مكون من 8 خانات ، ويمكن أن يكون للبيانات الحقيقية عرض قياسي للحقل ، مكون من 14 خانة ، منها 7 خانات على يمين العلامة العشرية . ويمكن أن يكون للبيانات بوليان عرض قياسي مكون من 6 خانات ، فإذا ما كان الحقل أعرض من اللازم ( كما هو الحال في العادة ) ، يوضع عنصر البيانات في الجزء الأيمن من الحقل . وينتج عن هذا وجود فراغ واحد أو أكثر يسبق البيانات كما هو موضح في المثال السابق .

ويمكن أن يتغير العرض القياسي للحقل بسهولة ليسمح للمبرمج بتحكم أكبر في شكل عناصر المخرجات . وطريقة عمل ذلك موضحة فيما بعد في هذا الفصل .

وعند كتابة سلسلة أو بيانات من النوع الحرفي كمخرجات ، يكون عرض الحقل مساويا بالضبط لعدد خانات عنصر البيانات . وعلى هذا ... فلن تكون هناك أى فراغات تسبق أو تتبع عنصر البيانات ، إلا إذا كانت هذه الفراغات جزءا من البيانات الفعلية .

مثال (٤-١١)

يحتوى أحد برامج البسكال على عبارة اكتب التالية :

```
write('RED',' WHITE',' BLUE');
```

سوف ينتج مايلي عن هذه العبارة

```
RED WHITE BLUE
```

لاحظ أن الفراغ الذي يفصل الكلمات الفردية موجود في آخر سلسلتين . أما إذا كانت عبارة اكتب على النحو التالي :

```
write('RED','WHITE','BLUE');
```

فسوف ينتج عنها مايلي :

```
REDWHITEBLUE
```

إذا ماتبع عبارة اكتب عبارة اكتب أخرى ، فتبدأ عبارة اكتب الثانية من عند انتهاء العبارة الأولى . وفي كلمات أخرى ... فإن أول عنصر بيانات يكتب بواسطة عبارة write الثانية يبدأ في السطر الحالي بعد آخر عنصر بيانات كتب بواسطة عبارة write الأولى مباشرة . وعلى هذا ... فلأليزم لعبارة write جديدة أن ينتج عنها سطر بيانات مخرجات جديد .

مثال (٤-١٢)

فيما يلي جزءاً من أجزاء برنامج بسكال

```
VAR a,b : real;
    i,j : integer;
    p,q : char;
    .
    .
    .
write(' a=',a,' b=',b,' i=',i);
write(' j=',j,' p=',p,' q=',q);
```

افرض أن القيم التالية حددت للمتغيرات :

Variable	Value
a	12500.0
b	-14.8
i	5
j	-9
p	X
q	Y

وعلى هذا تظهر بيانات المخرجات في سطر واحد على النحو التالي :

a= 1.2500000E+04 b=-1.4800000E+01 i= 5 j= -9 p=X q=Y

## 6. THE WRITELN STATEMENT

## ٦ - عبارة WRITELN :

عبارة writeln متطابقة مع عبارة write ، فيما عدا أنها ينتج عنها تحديد نهاية للسطر بعد كتابة آخر عنصر بيانات . وعلى هذا ... فإن عبارة write ، أو عبارة writeln تليها تبدأ في سطر جديد .

وتكتب العبارة على النحو التالي :

writeln(output data items)

حيث يمكن أن تكون عناصر بيانات المخرجات سلاسل أو ثوابت عددية ، أو قيم لمتغيرات أو تعبيرات من النوع الصحيح أو الحقيقي أو البولياني .

### مثال (٤-١٣)

افترض أن عبارات write الموجودة في المثال السابق استبدلت بعبارات writeln ، أي

```
VAR a,b : real;
    i,j : integer;
    p,q : char;
    .
    .
    .
writeln(' a=',a,' b=',b,' i=',i);
writeln(' j=',j,' p=',p,' q=',q);
```

وعلى هذا ينتج سطرًا المخرجات التاليان :

```
a= 1.2500000E+04 b=-1.4800000E+01 i=      5
j=      -9 p=X q=Y
```

لاحظ أن أول عبارة writeln هي التي تسببت في تجزئة السطر الفردي الأعلى للمخرجات إلى سطرين . أما ثاني عبارة write ، فليس لها أي تأثير على المخرجات ، بالرغم من أن أي مخرجات تالية subsequent تبدأ في سطر جديد . وعلى هذا ... فإن العبارتين التاليتين :

```
writeln(' a=',a,' b=',b,' i=',i);
write(' j=',j,' p=',p,' q=',q);
```

ينتج عنهما سطرًا المخرجات الذي سبق ذكرهما ، أما العبارتان التاليتان :

```
write(' a=',a,' b=',b,' i=',i);
writeln(' j=',j,' p=',p,' q=',q);
```

فينتج عنهما سطرًا المخرجات الذي سبق ذكره في المثال رقم ١٢ - ٤ .

ويمكن استخدام عبارة writeln فارغة empty ، وذلك لإنتاج سطر فارغ ، كما هو موضح في المثال التالي .

### مثال (٤-١٤)

اعتبر الثلاث عبارات writeln التالية :

```
writeln('line one');
writeln;
writeln('line two');
```

وينتج عن هذه العبارات ثلاثة أسطر مخرجات ( تشمل سطر فارغا ) ، كما هو موضح أدناه :

line one

line two

فإذا لم توجد عبارة writeln الفارغة ؛ فلن يظهر السطر الفارغ ، ويظهر السطران المطبوعان وراء بعضهما على النحو التالي :

line one

line two

## 7. FORMATTED OUTPUT

### ٧ - المخرجات المشكلة :

عادة ما يمكن جعل بيانات المخرجات مضبوطة بتغيير عرض الحقول المصاحبة للبيانات العددية ، وبيانات بوليان . ويمكن أن يتحقق ذلك بسهولة بإضافة معالم تشكيل format معينة داخل عبارات write ، وعبارات writeln ، وبصفة خاصة فإن كل عنصر حقيقي أو بوليان في عبارة write أو عبارة writeln يمكن أن تتبعه فاصلة ، أو كمية صحيحة موجبة تحدد عرض الحقل . ويمكن التعبير عن هذه الكمية ككثابت أو متغير أو تعبير .

مثال (٤-١٥)

يحتوي أحد برامج البسكال على العبارة التالية :

```
write('The answer is',sum : 4);
```

افترض أن sum متغير صحيح ، محدد له القيمة 16- وعلى ذلك تكتب القيمة باستخدام حقل عرضه 4 خانات ، يعطى المخرجات التالية :

The answer is -16

( لاحظ أن الفراغ الذي يسبق العدد ، هو جزء من عرض الحقل المكون من ٤ خانات ) .

مثال (٤-١٦)

افترض الآن أن عبارة write في المثال السابق أخذت الشكل التالي :

```
write('The answer is',sum : j+2);
```

فإذا كان j متغيراً صحيحاً ، محدد له القيمة 3 ، فإن عرض حقل النتيجة المصاحبة للمتغير sum يكون 5 خانات . وتظهر المخرجات على النحو التالي :

The answer is -16

( لاحظ أن هناك فراغين يسبقن العدد ) .

ويفسر الفرض المحدد للحقل بأنه أدنى minimum عرض حقل يصاحب عنصر البيانات . أما إذا كان الحقل كبيراً جداً ، فيرحل عنصر البيانات إلى يمين الحقل ، تاركا فراغا واحداً أو أكثر قبل العنصر . ومن ناحية أخرى ، يضاف فراغ إضافي بصورة تلقائية إذا ما كان الحقل صغيراً جداً . وهذا يمنع تبديل عناصر البيانات .



#### مثال (٤-١٧)

بالعودة إلى المثال رقم ٤ - ١٥ ، افترض أنه تحدد قيمة 3 للمتغير sum . وعلى هذا .. تظهر المخرجات كما يلي

The answer is 3

( لاحظ أن العدد محصور في الجزء الأيمن للحقل ، وينتج عن ذلك وجود ثلاثة فراغات بين is , 3 ) .

ومن ناحية أخرى .. إذا ماتحدد قيمة 10000 للمتغير sum ، فإن المخرجات تظهر على النحو التالي :

The answer is 10000

وقد أضيف فراغان إضافيان لحقل المخرجات ، ليلائم عدداً من 5 خانات ، والإشارة التي تسبقه .

اعتبر الآن أنه تحدد قيمة -30000 للمتغير sum . بهذا تصبح المخرجات على النحو التالي :

The answer is -30000

ويمكن كتابة عدد من النوع الحقيقي بطريقة مريحة أكثر ( بدون الأس ) إذا ماكان هناك عنصر مشكل ثانٍ . وعلى هذا ... فإن تحديد التشكيل الكامل لعدد حقيقي ، يكون عبارة عن فاصلة يتبعها رقم صحيح يحدد إجمالى عرض الحقل ، يتبعه فاصلة أخرى ، ويتبع هذه الفاصلة رقم صحيح آخر ، يحدد عدد المواقع الموجودة على يمين العلامة العشرية . وسوف يقرب الكسر العشري للعدد ذاتياً عندما تكون هناك حاجة لذلك .

#### مثال (٤-١٨)

افترض أن root متغير حقيقي ، محدد له القيمة 17487.266 ، يمكن كتابة هذا العدد بعدة طرق مختلفة كما هو موضح أدناه :

Writeln Statement	Resulting Output
writeln('root=',root);	root= 1.7487266E+04
writeln('root=',root : 12);	root= 1.74873E+04
writeln('root=',root : 18);	root= 1.74872660000E+04
writeln('root=',root : 10 : 3);	root= 17487.266
writeln('root=',root : 8 : 1);	root= 17487.3

تحتوى لغة البسكال على عبارة صفحة page . وهى عملية قياسية تتسبب فى ظهور عنصر البيانات المطبوع التالى فى قمة صفحة جديدة .

#### مثال (٤-١٩)

افترض أن أحد برامج البسكال يحتوى على الثلاث عبارات التالية :

```
writeln(a,b,c);
page;
writeln(x,y,z);
```

تتسبب هذه العبارات في طباعة قيم المتغيرات a , b , c في سطر واحد ، وتتبعها قيم المتغيرات x , y , z في سطر آخر . وتتسبب عبارة page في طباعة سطر المخرجات الثاني في بداية صفحة جديدة .

ولتمييز العديد من النظم المتداخلة عبارة page ( فهي تهمل ببساطة ) ، إلا أن عبارة page تتسبب - على أية حال - مع بعض النظم المتداخلة في مسح محتويات الشاشة .

وتفسر بعض نظم الكمبيوتر أول خانة من كل سطر من أسطر المخرجات بأنها خانة التحكم في الطابع printer control . ولاتطبع محتويات هذه الخانة ، لكنها تستخدم للتحكم في المسافات الموجودة بين أسطر الطباعة .

وفيما يلي رموز تحكم شائعة الاستخدام في التحكم في الطابع

الرمز	تفسيره
فراغ	الطباعة تحدث بعد السطر السابق مباشرة ( مخرجات فردية المسافة بين الأسطر )
0	يترك سطر فارغ ، ويكتب في السطر التالي له ( مخرجات زوجية المسافة بين الأسطر )
1	يترك صفحة فارغة قبل البدء في الطباعة .
+	تتم الطباعة على نفس السطر السابق .

مثال (٤-٢٠)

يشمل أحد برامج البسكال عبارات writeln التالية :

```
writeln(a,b,c);
writeln('1',x,y,z);
```

تتسبب هذه العبارات في طباعة قيم المتغيرات a , b , c على سطر واحد ، تتبعها قيم المتغيرات x , y , z على سطر آخر . ويظهر السطر الثاني في بداية صفحة جديدة - على أية حال - وذلك بسبب وجود رمز التحكم في الطابع في عبارة writeln الثانية . وعلى هذا ... فسوف تظهر المخرجات بنفس الطريقة التي ظهرت بها في المثال السابق .

أما إذا ما تغيرت عبارة writeln إلى الشكل التالي :

```
writeln('0',x,y,z);
```

فيظهر سطر المخرجات في نفس الصفحة ، مع وجود سطر فارغ بينهما ( تكون المخرجات مزبوجة المسافة ) .

ويجب أن يفهم أن هذه الرموز المستخدمة في التحكم في الطابع لاتميزها كل صيغ البسكال . ويجب أن يحدد القارئ ما إذا كانت هذه السمة ميسرة له في الجهاز المتاح له أم لا .

## 8. CLOSING REMARKS

٨ - ملاحظات أخيرة :

قبل ترك هذا الفصل ، توجد هناك ملاحظات عامة يمكن ذكرها عن عبارات page , writeln , write , readln

, , read

أولاً : هذه هى فى الواقع إجراءات قياسية standard procedures موجودة كجزء من لغة البسكال . وقد ناقشنا بالفعل استخدام الإجراءات القياسية المختلفة ، وطريقة الاتصال بها فى الفصل الثانى من الكتاب . وعلى القارئ أن يتذكر مرة أخرى أن الاتصال بإجراء قياسى يعتبر نوعاً من أنواع العبارات البسيطة . وفى هذا المفهوم تشير إلى استخدام هذه الإجراءات كمعبارات .

ثانياً : يجب أن يكون مفهوماً أن هذه العبارات تمثل فى الواقع فئة جزئية بسيطة من إمكانات المدخلات والمخرجات فى البسكال . وسوف نعتبر موضوع عمليات المدخلات والمخرجات بتفصيل أكبر عند دراسة الملفات فى الفصل الحادى عشر من الكتاب .

وأخيراً يجب أن يقدر القارئ أن هذه العمليات للمدخلات والمخرجات ، بالإضافة إلى المادة التى سبق ذكرها فى الفصول السابقة تقدم لنا إمكانية كتابة برامج بسكال كاملة ، لكنها بسيطة . وسوف يحتوى الفصل القادم على تعليمات لتنظيم وكتابة وتشغيل مثل هذه البرامج .

## Review Questions

## أسئلة للمراجعة :

- (١) ماهو ملف المدخلات ؟ كيف يتم إدخال عناصر البيانات فى ملف المدخلات ؟
- (٢) ماهو ملف المخرجات ؟ ماذا يحدث لعناصر البيانات التى يتم إدخالها فى ملف المخرجات ؟
- (٣) كيف ترتب عناصر البيانات داخل ملف المدخلات أو ملف المخرجات ؟
- (٤) كيف يشار إلى ملف المدخلات وملف المخرجات فى برنامج البسكال ؟
- (٥) ماهو الغرض من عبارة اقرأ read ؟
- (٦) فى أى معنى يجب أن تناظر عناصر البيانات فى ملف المدخلات متغيرات فى عبارة read ؟
- (٧) لخص قواعد وضع عناصر البيانات العددية والحرفية فى ملف المدخلات .
- (٨) هل يمكن تشغيل بيانات بوليان بعبارة read ؟
- (٩) كيف تقرأ عناصر البيانات من ملف مدخلات عندما تتبع عبارة read عبارة أخرى ؟
- (١٠) ماهو الغرض من عبارة readln ؟ وماهو الفرق بينها وبين عبارة read ؟
- (١١) ماذا يحدث عندما تسبق عبارة readln عبارة read ؟ وماذا يحدث إذا ماسبقت عبارة readln عبارة أخرى ؟
- (١٢) ماهو الغرض من دالة eof ؟ وماهو نوع المعلومات الذى تقدمه هذه الدالة ؟
- (١٣) ماهو الغرض من دالة eof ؟ وماهو نوع المعلومات الذى تقدمه هذه الدالة ؟ وماهو الفرق بين هذه الدالة ودالة eofln ؟
- (١٤) فى كلمات بسيطة ... كيف يمكن استخدام دالتى eof , eofln لقراءة عدد غير محدد من عناصر البيانات من ملفات مدخلات .
- (١٥) ماهو الغرض من عبارة write ؟ وماهو نوع عناصر المخرجات التى يمكن أن توجد فى هذه العبارة ؟
- (١٦) هل يمكن أن توجد سلاسل داخل عبارة write ؟ وهل يمكن أن توجد بها ثوابت عديدة ؟ وهل يمكن أن توجد بها تعبيرات ؟ قارن ذلك بعبارة read .

- (١٧) كيف تظهر الأعداد الحقيقية في العادة عندما تكتب بواسطة عبارة write ؟
- (١٨) هل يمكن تشغيل بيانات من نوع بوليان بعبارة write ؟ قارن ذلك بعبارة read .
- (١٩) ماذا يعنى عرض الحقل ؟ وماهو عرض الحقل النمطى المستخدم فى الكمبيوتر المتاح لك ؟
- (٢٠) لخص القواعد المستخدمة فى تحديد أماكن عناصر بيانات المخرجات .
- (٢١) كيف تكتب عناصر البيانات فى ملف المخرجات عندما تتبع عبارة write عبارة write أخرى ؟
- (٢٢) ماهو الغرض من عبارة writeln ؟ وماهو الفرق بينها وبين عبارة write ؟
- (٢٣) ماذا يحدث عندما تسبق عبارة writeln عبارة write ؟ وماذا يحدث عندما تسبق عبارة writeln عبارة writeln أخرى ؟
- (٢٤) لماذا يمكن استخدام عبارة writeln فارغة فى برنامج البسكال ؟ أى نوع من أنواع العبارات التى يتبعها عبارة writeln الفارغة فى معظم الأحوال ؟
- (٢٥) كيف يمكن تغيير عرض الحقل فى ملف المخرجات ؟ وعلى أى نوع من أنواع البيانات يحدث ذلك ؟
- (٢٦) ماذا يحدث إذا كان أحد حقول المخرجات أعرض من اللازم بالنسبة لعنصر بيانات معين ؟ وماذا يحدث اذا ماكان الحقل أقل من اللازم ؟
- (٢٧) كيف يظهر عدد حقيقى فى صورة عشرية قياسية ( أى بدون أس ) ؟
- (٢٨) افرض أن أحد الأعداد الحقيقية يظهر فى غير الصورة العلمية ، ولم يكن الحقل عريضاً بدرجة تكفى لكسر العشرى للعدد . هل سيحدث تقريب للكسر ، أم سيحدث حذف دون تقريب ؟
- (٢٩) ماهو الغرض من عبارة page ؟ كيف تفسر بعض نظم الكمبيوتر المتداخلة هذه العبارة ؟
- (٣٠) هل يمكن استخدام رموز التحكم فى الطابع على جهاز الكمبيوتر المتاح لك ؟ إذا كانت الإجابة بنعم ؛ فما هى الرموز المتاحة ؟ وماهو الغرض من كل رمز من هذه الرموز ؟
- (٣١) هل وصف هذا الفصل كل إجراءات المدخلات والمخرجات الموجودة فى البسكال ؟ وضح ذلك .

## Solved Problems

## مسائل محلولة :

- (٣٢) فيما يلى هيكلاً تخطيطياً لبرنامج بسكال ، مع التركيز على معالم المدخلات والمخرجات

```
PROGRAM sample(input,output);
CONST factor = 12345;
      flag = 'entry point';
VAR i1,i2 : integer;
      r1,r2,r3 : real;
      c1,c2 : char;
      b1 : boolean;
```

```

BEGIN
.
.
.
readln(i1,i2,r1,r2,c1);
.
.
.
writeln(' I1=',i1:4, ' I2=',i2:4);
writeln;
writeln(' R1=',r1:10:2, ' R2=',r2:10:2, ' R3=',r3:12:4);
writeln;
write(' FIRST CHAR IS ',c1,' SECOND CHAR IS ',c2);
write(' TEST STATUS: ',b1:5);
.
.
.
END.

```

سوف يقرأ هذا البرنامج سطرا واحداً من البيانات ( فيه خمسة عناصر بيانات ) من ملف مدخلات ، وسوف يكتب البرنامج ثلاثة أسطر بيانات في ملفات المخرجات . وسوف يتم تسمية وتشكيل عناصر بيانات المخرجات ، كما أن المسافات بين أسطر المخرجات تكون مزبوجة .

(٣٣) تشير مواقف المشاكل التالية الى ثوابت ومتغيرات معرفة في المسألة السابقة . اكتب عبارات مناسبة ، أو مجموعة عبارات لكل موقف من هذه المواقف .

(١) اقرأ قيم المتغيرات i1 , r3 , c2 من سطر بيانات واحد

```

read(i1,r3,c2);
or
readln(i1,r3,c2);
or
read(i1);
read(r3);
read(c2);

```

(ب) اقرأ قيم المتغيرات c1,r2,r1,i1 من سطر واحد ، تتبعه قيم المتغيرات i2,r3,c2 من سطر آخر ،

```

readln(i1,r1,r2,c1);
readln(i2,r3,c2);

```

يمكن استبدال ثانى عبارة readln بعبارة read على النحو التالي :

```

read(i2,r3,c2);

```

(ج) اكتب قيم الثابتين على سطر واحد ، تليه قيم كل المتغيرات على سطر آخر . لاتضع أسماء للمخرجات ، واترك فراغا بين الثابتين . اترك سطرين فارغين بين سطرى المخرجات ( أى مسافة ثلاثية ) .

```
writeln(factor, ' ', flag);
writeln;
writeln;
writeln(i1, i2, r1, r2, r3, c1, c2, b1);
```

يمكن استبدال آخر عبارة writeln بعبارة write التالية :

```
write(i1, i2, r1, r2, r3, c1, c2, b1);
```

(د) اكتب قيم الثابتين على سطر واحد ، تليه قيم c1, r2, r1, i1 على سطر آخر ، يليه قيم c2, b1, r3, i2 على سطر آخر . ابدأ فى بداية صفحة جديدة ، مع ترك مسافة مزدوجة بين أسطر المخرجات . ضع أسماء لكل عناصر المخرجات .

```
page;
writeln('Factor=', factor, ' Flag=', flag);
writeln;
writeln('I1=', i1, ' R1=', r1, ' R2=', r2, ' C1=', c1);
writeln;
writeln('I2=', i2, ' R3=', r3, ' B1=', b1, ' C2=', c2);
```

إذا ما استخدمت إحدى صيغ النسخال الموجود بها رموز التحكم فى الطابع ، فيمكن كتابة العبارات السابقة على النحو التالى أيضا :

```
writeln('Factor=', factor, ' Flag=', flag);
writeln('OI1=', i1, ' R1=', r1, ' R2=', r2, ' C1=', c1);
writeln('OI2=', i2, ' R3=', r3, ' B1=', b1, ' C2=', c2);
```

(هـ) أعد المسألة السابقة ، ولا تضع فى هذه الحالة أسماء لبيانات المخرجات ، ولكن شكل بيانات المخرجات على النحو التالى :

integer: 5 characters

real: 10 characters, with 3 digits to the right of the decimal point (no exponent)

boolean: 7 characters

كما هو مطلوب أن يسبق كل عنصر بيانات حرفى فراغ واحد

```
page;
writeln(factor:5, ' ', flag);
writeln;
writeln(i1:5, r1:10:3, r2:10:3, ' ', c1);
writeln;
writeln(i2:5, r3:10:3, b1:7, ' ', c2);
```

يمكن إعادة كتابة هذه العبارات باستخدام خانات تحكم الطابعة كما يلى

```
writeln('1', factor:5, ' ', flag);
writeln('0', i1:5, r1:10:3, r2:10:3, ' ', c1);
writeln('0', i2:5, r3:10:3, b1:7, ' ', c2);
```

(٣٤) افرض أن ملف المدخلات يحتوى على سطرى البيانات التاليين :

10 0.005 4.66E12ABC

-817 2.7E-3 XYZ

ماهى القيم التى تحدد للمتغيرات الموجودة فى السؤال السابق (ب) .

```
i1=10  r1=0.005  r2=4.66E+12  c1=A
i2=-817  r3=2.7E-03  c2= (blank space)
```

(٣٥) افرض أن أحد ملفات المدخلات يحتوى على الأربعة أسطر بيانات التالية :

```
10  0.005
4.66E12  ABC  -63  17.7
-75  33.9E  B
C
```

ماهى القيم التى تحدد للمتغيرات الموجودة فى السؤال الثانى (ب) ؟

```
i1=10  r1=0.005  r2=4.66E+12  c1= (blank space)
i2=-75  r3=33.9  c2=E
```

(٣٦) افرض أن المتغيرات الموجودة فى السؤال (٣٤) محدد لها القيم التالية :

Variable	Value
i1	-630
i2	375
r1	20.8
r2	-477300.0
r3	0.000185
c1	\$
c2	5
b1	خطا

بين كيف تظهر البيانات فى ملف المخرجات إذا ما كتبت طبقا لعبارات writeln المعطاه فى السؤال الثانى (ج) .

```
12345 entry point
(blank line)
(blank line)
-630  375 2.0800000E+01-4.7730000E+05 1.8500000E-04$5 false
```

(٣٧) أعد المسألة السابقة ، مفترضا أن بيانات المخرجات مكتوبة طبقا لعبارات writeln المعطاه فى السؤال الثانى (د) .

```
Factor=12345 Flag=entry point
(blank line)
I1=  -630 R1= 2.0800000E+01 R2=-4.7730000E+05 C1=$
(blank line)
I2=  375 R3= 1.8500000E-04 B1= false. C2=5
```

(٣٨) أعد المسألة رقم ٣٦ ، مفترضا أن بيانات المخرجات مكتوبة طبقا لعبارات writeln المعطاه فى السؤال ٣٤ (هـ) .

```
12345 entry point
(blank line)
-630  20.800-477300.000 $
(blank line)
375  0.000 false 5
```

(٣٩) فيما يلي جزءاً من برنامج بسكال

```
PROGRAM sample2(input,output);
VAR a,b,c,d : integer;
BEGIN
    .
    .
    .
    readln(a,b,c,d);
    .
    .
    .
    writeln(5*(a+b)/2 : c+2, 10*(a-b)/2 : c+d-1);
    .
    .
    .
END.
```

افرض أن ملف المدخلات يحتوى على البيانات التالية :

3 5 4 1

ماهى المخرجات التى تنتج عن هذا البرنامج ؟

القيمتان العدديتان هما 20 , -10 وعرض الحقل المناظر لكل منهما هو 6 , 4 على التوالى . وعلى هذا تظهر المخرجات كمايلى :

20 -10

مع وجود 4 فراغات تسبق أول عدد



## Supplementary Problems

## مشاكل متكاملة :

(٤٠) فيما يلي هيكلاً تخطيطياً لبرنامج بسكال كامل :

```

PROGRAM example(input,output);
CONST flag = 'red';
      factor = 0.005;
VAR i1,i2,i3 : integer;
    r1,r2 : real;
    c1,c2,c3,c4 : char;
    b1,b2 : boolean;
BEGIN
    .
    .
    .
    read(i1,i2);
    readln(i3,r1,r2);
    read(c1,c2,c3,c4);
    .
    .
    .
    page;
    writeln(flag,factor);
    writeln;
    write(i1,i2);
    writeln(i3,r1,r2);
    writeln;
    write(c1,c2,c3,c4);
    writeln(b1,b2);
    .
    .
    .
END.

```

في كلمات بسيطة ، كيف يجب إدخال عناصر بيانات المدخلات ؟ وكيف تبدو عناصر بيانات المخرجات ؟ هل تفصل عناصر المخرجات وتميز من بعضها ؟

(٤١) كل من المشاكل التالية يبين فئة من البيانات داخل ملف مدخلات . حدد في كل حالة القيم التي تحدد المتغيرات الموجودة في السؤال السابق .

(a) 1 2 3 4.0 5.0

blue green

(b) 1

2

3

4

5

b

1

u

e

(c) 1 2 3 4.0 5.0 blue

(d) 1 2

3 4

5 blue

green

(٤٢) كيف تظهر المخرجات الناتجة من السؤال الأول ، مفترضا أن القيم التالية محددة للمتغيرات ؟

Variable	Value
i1	100
i2	-200
i3	-300
r1	400.444
r2	-500.555
c1	P
c2	I
c3	N
c4	K
b1	true
b2	false

(٤٣) افرض أن عبارات المخرجات في السؤال الأول استبدلت بالعبارات التالية :

```
page;
writeln('flag=',flag,' factor=',factor:6:3);
writeln;
write('i1=',i1:4,' i2=',i2:4);
writeln('i3=',i3:4,' r1=',r1:6:1,' r2=',r2:6:1);
writeln;
write('color=',c1,c2,c3,c4);
writeln('b1=',b1:5,' b2=',b2:5);
```

كيف تبدو المخرجات ، مفترضا أن القيم الموجودة في السؤال السابق حددت للمتغيرات ؟

(٤٤) افرض الآن أن صيغة البسكال المستخدمة تستخدم رموز تحكم في الطابع ، وأن مخرجات السؤال الأول استبدلت بما يلي :

```
writeln('lflag=',flag,' factor=',factor:6:3);
writeln('0i1=',i1:5,' i2=',i2:5,' i3=',i3:5);
writeln(' r1=',r1:8:2,' r2=',r2:8:2);
writeln('lcolor=',c1,c2,c3,c4);
writeln('0b1=',b1:6,' b2=',b2:6);
```

كيف تبدو المخرجات ، مفترضا أن القيم الموجودة في السؤال الثالث حددت للمتغيرات ؟

(٤٥) مواقع المشاكل التالية تشير إلى ثوابت ومتغيرات معرفة في السؤال الأول . اكتب العبارات المناسبة ، أو مجموعة العبارات لكل من هذه المواقع .

(أ) اقرأ كل عناصر بيانات المدخلات من سطر بيانات واحد .

(ب) اقرأ قيم i1 , i2 , i3 , c4 من سطر بيانات واحد .

(ج) اقرأ عناصر البيانات الصحيحة من سطر بيانات واحد ، وعناصر البيانات الحقيقية من سطر آخر ، وعناصر البيانات الحرفية من سطر آخر .

(د) اقرأ كل عنصر بيانات من سطر منفصل .

(هـ) اكتب قيم الثابتين وكل المتغيرات في سطر واحد . لاتضع أسماء للمخرجات ، ولكن اترك فراغا واحدا على الأقل بين كل عنصر بيانات .

(و) أعد المسألة السابقة ، مع تشكيل البيانات العددية وبيانات بوليان طبقا لما يلي :

الصحيح : 4 خانات .

الحقيقي : 8 خانات ، منها اثنان على يمين العلامة العشرية ( بدون استخدام أس ) .

البوليان : 6 خانات

(ز) أكتب قيمتي الثابتين على سطر واحد ، تتبعه قيم المتغيرات الحرفية ، ومتغيرات بوليان على سطر آخر ، تتبعه قيم المتغيرات العددية على سطر ثالث . لاتضع أسماء للمخرجات ، واترك فراغا واحدا على الأقل بين كل عنصرى بيانات . اترك سطرًا فارغًا أيضا بين كل سطر من أسطر الطباعة ( أى أن الطباعة تكون مزدوجة المسافة ) .

(ح) أعد المسألة السابقة ، مع تسمية كل ثابت ، وكل قيمة عددية ، وكل عنصر بيانات بوليان . اكتب البيانات الحرفية كرموز متتالية ، دون ظهور أى فراغات مع ظهور اسم واحد يسبق هذه الرموز .

(ط) اكتب قيم flag , i1 , r1 , c1 , c2 , b1 على سطر واحد ، تتبعه قيم factor , i2 , i3 , r2 , c3 , c4 , b2 على سطر آخر . لاتضع أسماء للمخرجات ، وضع فراغ واحد على الأقل بين كل عنصرى بيانات . ابدأ أول سطر فى بداية صفحة جديدة ، واترك سطرين فارغين بين أسطر المخرجات المطبوعة .

(ى) أعد المسألة السابقة ، مع تشكيل البيانات العددية ، وبيانات بوليان على النحو التالى :

صحيح : 5 خانات

حقيقي : 7 خانات ، اثنان منها على يمين العلامة العشرية ( لا يوجد أس )

بوليان : 7 خانات .

(ك) اكتب القيم لكل من الثابتين وكل المتغيرات . ابدأ فى بداية صفحة جديدة ، واكتب كل عنصر بيانات على سطر مستقل . ضع أسماء لكل عناصر البيانات . شكل البيانات على النحو التالى :

صحيح : 4 خانات

حقيقي : 12 خانة ، مستخدما أس

بوليان : 5 خانات

(٤٦) فيما يلى جزءا من برنامج بسكال :

```
PROGRAM example2(input,output);
VAR w,x,y,z : integer;
BEGIN
.
.
.
readln(w,x,y,z);
.
.
.
writeln('SUM=',x+y+z : w, ' PRODUCT=',x*y*z : w+3);
.
.
.
END.
```

افرض أن ملف المدخلات يحتوى على القيم التالية :

3    10    20    30

بين كيف تكون المخرجات .



## الفصل الخامس

### إعداد وتشغيل برنامج بسكال كامل

## Preparing and Running a Complete Pascal Program

حتى الآن تعلمنا مايكفى من لغة البسكال لكتابة برنامج كامل وبسيط . وعلى هذا ... فلنلق بعض الانتباه على تخطيط وكتابة وتنفيذ مثل هذه البرامج . كما نأخذ فى الاعتبار أيضا أوجه البرمجة الخاصة بالبسكال ، مثل التطوير المنطقى للبرنامج ، وأسلوب البرمجة الجيد ، وعملية تطوير برنامج كامل .

### ١ - تخطيط برنامج البسكال : 1. PLANNING A PASCAL PROGRAM

من الضرورى رسم سياسة عامة للبرنامج قبل أن تبدأ أى تفاصيل برمجة فعلية . وبهذه الطريقة يستطيع أن يركز المبرمج أساسا على المنطق العام للبرنامج ، دون أن يدخل فى التفاصيل التكوينية للتعليمات الفردية . وقد تعاد عملية التخطيط الشامل هذه عدة مرات ، مع إضافة تفاصيل برمجة فى كل مرة تحدث فيها الإعادة . ويستطيع المبرمج على هذا أن ينقل انتباهه تدريجيا من سياسة الحسابات الشاملة إلى تفاصيل التعليمات الفردية . ويشار إلى مثل هذا الأسلوب بالبرمجة من القمة إلى القاعدة .

وعادة مايتم تنظيم البرنامج من القمة إلى القاعدة عن طريق إعداد تخطيط غير رسمى ، يحتوى على عبارات ( أو جمل ) كجزء من اللغة الإنجليزية وجزء من البسكال . وفى المرحلة الأولية تكون كمية البسكال قليلة جدا ، ولا تظهر سوى كلمات أساسية متعددة تعرف مكونات البرنامج الأساسية فقط ، مثل ( END , BEGIN , PROGRAM ) . وبعد ذلك تدخل المادة الوصفية باللغة الإنجليزية بين هذه الكلمات الأساسية . ويكون ذلك عادة على هيئة تعليقات فى البرنامج . وعادة مايشار إلى التخطيط الناتج بأنه شفرة شبيهة pseudo code .

مثال (٥-١)

بيع الفطائر Pizza : يبيع أحد محلات بيع الفطائر ثلاثة أحجام من الفطائر ، وهى الحجم الصغير ( قطره 10 بوصة ) ، والحجم المتوسط ( قطره 12 بوصة ) ، والحجم الكبير ( قطره 16 بوصة ) . ويمكن شراء الفطيرة بدون إضافات ، أى بالجبن والصلصة فقط ، أو بإضافات مثل البصل والجمبرى وغيرها .

ويريد صاحب المحل أن يعد برنامج كمبيوتر يحسب سعر بيع الفطيرة إذا ما أعطى له حجمها والمكونات الإضافية . سعر البيع يساوى مرة ونصف إجمالى التكلفة ، طبقا لحجم الفطيرة وعدد المكونات الإضافية . وسوف يشمل إجمالى التكلفة تكلفة أعداد ثابتة ، وتكلفة متغيرة تتناسب مع حجم الفطيرة ، وتكلفة متغيرة إضافية لكل مكون إضافى زيادة . ويفترض التبسيط ، يفترض أن تكلفة كل مكون إضافى لوحدة المساحة متساوية .

اعتبر الآن الفطيرة التى قطرها d وبها n مكون إضافى . يحسب سعر بيع هذه الفطيرة على النحو التالى :

$$\text{price} = 1.5 * \text{cost}$$

$$\text{cost} = \text{fixedcost} + (\text{basecost} * \text{area}) + (n * \text{extracost} * \text{area})$$

حيث

$$\text{area} = \frac{\pi d^2}{4}$$

و

وعلى هذا يمكن تحديد سعر بيع الفطيرة ببساطة إذا كانت التكاليف المختلفة معروفة ، وكذلك يكون القطر وعدد المكونات الإضافية معروفين أيضا .

لاحظ أنه يمكن قراءة مكونات التكلفة مع بداية كل حالة ، وذلك مع القطر وعدد المكونات الإضافية . كما يمكن أيضا تعريف مكونات التكلفة كثوابت داخل البرنامج . وسوف نتبع الطريقة الأخيرة . وسبب ذلك هو الافتراض أن البرنامج سوف ينفذ العديد من المرات باستخدام قيم مختلفة للمدخلات بالنسبة للقطر وعدد المكونات الإضافية مع استخدام نفس التكاليف .

يمكننا الآن كتابة التخطيط العام التالي للبرنامج :

- (١) تعريف أو تحديد عناصر التكلفة المختلفة .
  - (٢) قراءة القطر للفطيرة وعدد المكونات الإضافية .
  - (٣) حساب مساحة الفطيرة .
  - (٤) حساب التكلفة الكلية ، وسعر البيع .
  - (٥) كتابة الإجابة النهائية ( سعر البيع ) مع كتابة مدخلات كافية لتعريف المشكلة .
- فإذا ما كتبنا هذا التخطيط بالشجرة الشبيهة ، فيمكننا الحصول على مايلي :

(\* define the cost components as program constants \*)

BEGIN

(\* read the diameter and the number of extra ingredients \*)

(\* calculate the area of the pizza, the overall cost and the selling price \*)

(\* write out the input data and the selling price \*)

END.

وحيث إن هذه المشكلة بسيطة بصفة خاصة ، فلا تكون هناك حاجة إلى إدخال تعديلات إضافية على هذا التخطيط . أما إذا كانت المشكلة أكثر تعقيدا ، فيجب أن نكتب صيفا تفصيلية أكثر بالشجرة الشبيهة ، موضحين تفاصيل أدق للمنطق الشامل للبرنامج .

وهناك طريقة أخرى تستخدم في بعض الأحيان في تخطيط برنامج البسكال ، وهي أسلوب : من القاعدة إلى القمة . ويمكن أن تكون هذه الطريقة مفيدة في حالة البرامج التي تستخدم برامج فرعية داخل البرنامج ( أى تستخدم دوال وإجراءات يعدها المستفيد ) والتي يمكن الاتصال بها من أجزاء أخرى من أجزاء البرنامج . وفي أسلوب من القاعدة إلى القمة يتم تطوير هذه البرامج الفرعية للبرنامج بالتفصيل أولا ، وذلك في مرحلة مبكرة من عملية التخطيط الشامل . ويمكن بعد ذلك أن يعتمد أى تطوير للبرنامج على الخواص المعروفة لهذه الأجزاء من البرنامج ، أو هذه البرامج الفرعية . وعادة ما يستخدم كلا من الأسلوبين عمليا : أسلوب من القاعدة إلى القمة لإعداد البرامج الفرعية قبل إعداد الجزء الرئيسى من البرنامج ، وأسلوب من القمة للقاعدة بالنسبة لإعداد كل برنامج فرعى . ويتم إعداد الجزء الرئيسى من البرنامج باستخدام أسلوب من القمة للقاعدة في معظم الأحوال .

## 2. WRITING A PASCAL PROGRAM

## ٢ - كتابة برنامج بسكال :

بعد صياغة سياسة عامة للبرنامج ، وكتابة تخطيط البرنامج ، يمكن توجيه الاتجاه إلى تفاصيل عمل برنامج البسكال . وعند هذه النقطة يصبح التركيز على ترجمة كل خطوة من خطوات تخطيط البرنامج ( أو كل جزء من أجزاء الشجرة الشبيهة ) إلى إحدى تعليمات البسكال ، أو عدة تعليمات بسكال . ويجب أن يكون هذا النشاط نشاطا مباشرا ، مع افتراض أن سياسة البرنامج العامة تم إعدادها بدقة وتفاصيل كافية .

وعند كتابة برنامج بسكال كامل يوجد - على أية حال - عدة نقاط يجب تذكرها دائما . أولا : يجب إدخال التوضيحات والتعريفات المختلفة بالترتيب المناسب . ثانيا : يجب إعطاء بعض الاهتمام بالسماوات التي سوف تكون متاحة للمستفيد ( بالرغم من أن بعض هذه السماوات يجب أن تؤخذ في الاعتبار في مرحلة تخطيط البرنامج ) . وأخيرا يجب الاهتمام بأوجه معينة لأسلوب البرمجة ، والتي تؤخذ في الاعتبار عند تنظيم البرنامج ليكون واضحا ومعبرا . وناخذ كل من هذه النقاط بشئ من التفصيل .

سبق أن قدمنا في الفصل الأول تخطيطا يوضح الهيكل الشامل لبرنامج البسكال . وفيما يلي إعادة لهذا التخطيط ، يليه بعض التعليقات التوضيحية :

1. Header
2. Block
  - (a) Declarations
    - Labels
    - Constants
    - Type definitions
    - Variables
    - Procedures and functions
  - (b) Statements

تذكر أن العنوان عبارة عن عنصر من سطر واحد ، يبدأ بكلمة PROGRAM ، ويتبعه اسم البرنامج وأدلة لأى ملفات مدخلات أو مخرجات ، وتوضع أدلة الملفات بين قوسين ، وتفصل عن بعضها بواسطة فواصل .

وتحتوى كتلة البرنامج ( المجموعة الرئيسية للبرنامج ) على جزء توضيحات ، وجزء لعبارات إجراءات ، ويجب تقديم التوضيحات بالترتيب المطلوب ، وذلك بالرغم من أن أنواع التوضيحات المختلفة ليست فى حاجة إلى أن تظهر فى كل برنامج من برامج البسكال . وفى واقع الأمر ، فإن الثوابت والمتغيرات هى أنواع التوضيحات الوحيدة التى تؤخذ فى الاعتبار حتى الآن فى برنامج البسكال . أما النقطة الهامة الآن ، فهى أن نتذكر أن تعريفات الثوابت يجب أن تسبق توضيحات المتغيرات .

ومن الناحية التكوينية ، يسمح بعمق كبير فى كتابة عبارات الإجراءات . والمتطلب الثابت الوحيد هو أن توجد عبارات الإجراءات داخل عبارة شاملة مركبة ( أى بين BEGIN , END ) . وعلى هذا ... فيجب أن يحتوى كل برنامج على تسلسل BEGIN ، تليه بقية العبارات ، وينتهى بكلمة END . ويمكن أن توجد بقية العبارات المركبة داخل هذه العبارة المركبة الشاملة إذا كانت هناك رغبة فى ذلك . ويمكن أن تحتوى كل عبارة مركبة على عبارات إجراءات مختلفة ، أو على مجموعات عبارات ( وهناك المزيد عن ذلك فى الفصل القادم ) .

يجب إعطاء بعض الاهتمام بأمر التنقيط ، وبصفة خاصة باستخدام الفواصل المنقوطة . وتستخدم القواعد التالية فى تحقيق ذلك .

- (١) تستخدم الفاصلة المنقوطة فى البسكال كفاصل Separator ( وايس كمحدد لنهاية terminator ) . وعلى هذا فإنها تستخدم بين عبارتين متتاليتين أو توضيحين متتالين ، بدلا من استخدامها فى نهاية كل عبارة أو كل توضيح .

(٢) تستخدم قواعد خاصة مع BEGIN , END , فهذان العنصران هما فى واقع الأمر قوسان brackets يحددان بداية ونهاية عبارة مركبة . وعلى هذا ... فلا تحتاج BEGIN لأن تليها فاصلة منقوطة ، كما لا تحتاج END لأن تسبقها فاصلة منقوطة .

(٣) سوف تفسر الفاصلة المنقوطة غير المطلوبة ( مثل التى تسبق END ) بأنها عبارة صفرية null statement ( أى عبارة خالية ) .

وعادة ما لا يكون لهذا تأثير ملحوظ على تنفيذ البرنامج ، بالرغم من أنه هناك مواقف معينة يمكن أن يتغير فيها منطق البرنامج ... دون أن يكون هذا مطلوبا . وعلى هذا ... فيجب تجنب ظهور الفواصل المنقوطة غير المطلوبة .

(٤) يجب ان ينتهى كل برنامج كامل بنقطة . وعلى هذا ... فان آخر END فى البرنامج يجب أن يتبعها نقطة .

ورسومات بسكال التكوينية الموجودة فى نهاية هذا الكتاب يمكن فحصها للحصول على معلومات دقيقة عن استخدام التنقيط مع عبارات محددة .

#### مثال (٥-٢)

بيع الفطائر : المرضى أن مشكلة تسعير الفطائر المذكورة فى المثال السابق تستخدم بيانات التكلفة التالية :

- تكلفة ثابتة = \$ 0.75 لكل فطيرة
- تكلفة أساسية = \$ 0.01 لكل بوصة مربعة .
- تكلفة كل عنصر إضافي = \$ 0.0025 لكل بوصة مربعة .

وفيما يلى برنامج بسكال أولى يحتوى على عناصر التكلفة هذه :

```
PROGRAM pizzas(input,output);
CONST pi = 3.14159;
      fixedcost = 0.75;
      basecost = 0.01;
      extracost = 0.0025;
VAR n : integer;
     d,area,cost,price : real;
BEGIN
  readln(d,n);
  area := pi*sqr(d)/4;
  cost := fixedcost + basecost*area + n*extracost*area;
  price := 1.5*cost;
  writeln(d,n,price)
END.
```

لاحظ أن البرنامج يحتوى على تعريفات ثوابت وتوضيحات متغيرات ، وتسبق تعريفات الثوابت توضيحات المتغيرات . لاحظ أيضا أن التعريفات والتوضيحات والعبارات مفصولة بواسطة فواصل منقوطة ، مع ظهور الفواصل المنقوطة فى نهاية كل سطر عندما تكون هناك حاجة لذلك . ولاتتبع كلمة BEGIN فاصلة منقوطة ، كما لاتسبق كلمة END فاصلة منقوطة ، حيث إن هاتين الكلمتين الأساسيتين تخدمان كاقواس تعرف بداية ونهاية عبارة مركبة واحدة . لاحظ أخيرا النقطة الموجودة فى نهاية البرنامج كما هو مطلوب تماما .

والبرنامج السابق هو برنامج بسكال كامل ، إلا أنه ينقصه بعض السمات المرغوب فيها . وسوف تقدم صيغتان أكثر اكتمالا ( أكثر رغبة فى الحصول عليهما ) فى المثالين التاليين .



يجب أن يفهم القارئ أنه هناك المزيد لكتابة برنامج بسكال كامل عن ترتيب توضيحات وعبارة ببساطة في ترتيبها الصحيح ، ووضع الفواصل والنقط بطريقة صحيحة . يجب توجيه الاهتمام أيضا إلى وجود سمات خاصة تحسن من قراءة البرنامج ، ومن نتائج مخرجاته . وتشمل هذه السمات التسلسل المنطقي للعبارة ، واستخدام الترحيل ، واستخدام التعليقات ، وإنتاج مخرجات بأسماء مناسبة .

ويحدد تسلسل العبارات المنطقي داخل البرنامج إلى حد كبير ، وذلك بتحديد منطق البرنامج . وعادة ما يكون هناك - على أية حال - طرق مختلفة عديدة لتسلسل عبارات معينة ، نون تغيير في منطق البرنامج . وهذا صحيح بصفة خاصة بالنسبة للبرامج الأكثر تعقيدا ، والتي تحتوي على استخدام أجزاء برنامج شرطية أو متكررة . وفي مثل هذه الحالات يمكن أن يكون لتسلسل عبارات معينة أو مجموعة عبارات تأثير رئيسي على وضوح منطق البرنامج . ولذلك فمن المهم أن تكون العبارات متسلسلة في أفضل طريقة فعالة . وسوف نذكر المزيد عن ذلك في الفصل القادم عند مناقشة أنواع الشروط المختلفة ، ومعالج التكرار المتاحة في البسكال .

ويرتبط استخدام الترحيل اتصالا وثيقا بتسلسل مجموعات العبارات داخل البرنامج . وبينما يؤثر التسلسل على ترتيب تنفيذ مجموعة العمليات ، فإن الترحيل يوضح طبيعة الأجزاء الخاصة بعبارات فردية داخل المجموعة . ومميزات الترحيل واضحة ، حتى بالنسبة للبرامج البسيطة التي سبق ذكرها في هذا الكتاب . وسوف يصبح ذلك أكثر وضوحا فيما بعد ، كلما أخذنا في الاعتبار برامج بسكال أكثر تعقيدا .

ويجب أن توجد تعليقات بصفة دائمة في برنامج البسكال . وإذا ما كتبت مثل هذه التعليقات بطريقة مناسبة ، فيمكنها أن تقدم نظرة عامة مفيدة للمنطق العام لبرنامج . كما يمكنها أيضا أن تخطط الأجزاء الرئيسية للبرنامج ، وأن تعرف عناصر رئيسية معينة داخل البرنامج ، وأن تقدم معلومات مفيدة أخرى عن البرنامج . ( والتعليقات من هذا النوع يمكن أن تكون مفيدة جدا للمبرمج ، ولأي شخص آخر يحاول أن يقرأ البرنامج ويفهمه ، حيث ينسى المبرمجون في بعض الأحيان تفاصيل برامجهم مع مرور الزمن . وهذا صحيح بصفة خاصة بالنسبة للبرامج الطويلة المعقدة ) . وعادة ما لا تحتاج التعليقات التي توضع داخل البرنامج أن تكون طويلة وكثيرة ، فقرة من التعليقات التي توضع في أماكن مناسبة يمكنها أن تلقى ضوئا كافيا على البرنامج .

وهناك صفة هامة أخرى للبرنامج الجيد ، وهي قدرته على إنتاج مخرجات واضحة ومقروءة . ويسهم عاملان في هذا الوضوح ، أولهما : هو تسمية بيانات المخرجات كما سبق ذكره في الفصل الرابع من الكتاب . وثانيهما : هو ظهور بعض بيانات المدخلات مع المخرجات ، بحيث إن كل فئة من فئات بيانات المدخلات ( إذا ما كان هناك أكثر من فئة واحدة ) يمكن تعريفها بدقة . وطريقة تحقيق ذلك تعتمد على البيئة التي ينفذ فيها برنامج البسكال .

ففي البيئة غير المتداخلة يجب أن تطبع كمية معينة من بيانات المدخلات ، وذلك بالإضافة إلى المخرجات المطلوبة . إذا كانت المدخلات كبيرة ، فيجب عرض عدد بسيط من العناصر الرئيسية لبيانات المدخلات تكفي لتعريف كل مشكلة خاصة ( أي لتعريف كل فئة من فئات بيانات المخرجات ) .

مثال ( ٥ - ٣ )

نقدم الآن صيغة مطورة من برنامج تسعير الفطائر الذي سبق ذكره في المثال السابق . وتشمل الصيغة الحالية تعريف تعليقات وإنتاج بيانات مخرجات لها أسماء ، ومرتبطة بطريقة واضحة أكثر من المخرجات التي سبق إنتاجها في المثال السابق ( لاحظ أن برنامج المثال السابق لا يوجد به ترحيل لعبارة ، كما أنه يتسبب في طباعة بيانات المدخلات مع المخرجات المحسوبة ) .

نفترض أن هذا البرنامج سوف ينفذ في بيئة غير متداخلة . وعلى هذا ... فإننا نستمر في طباعة بيانات المدخلات مع المخرجات المحسوبة ، وذلك لتعريف كل مشكلة ( في ما إذا كان البرنامج ينفذ بعدة فئات بيانات مدخلات مختلفة ) .

```

PROGRAM pizzas(input,output);

(* This program calculates the selling price of a pizza,
   given the diameter and the number of extra items. *)

(* Non-interactive version *)

CONST pi = 3.14159;
      fixedcost = 0.75;
      basecost = 0.01;
      extracost = 0.0025;
VAR n : integer;
     d,area,cost,price : real;

BEGIN (* action statements *)
  readln(d,n);
  area := pi*sqr(d)/4;
  cost := fixedcost + basecost*area + n*extracost*area;
  price := 1.5*cost;
  writeln(' Pizza size (diameter):', d:4:0, ' inches');
  writeln(' Number of extra ingredients:', n:2);
  writeln;
  writeln(' The selling price is: $', price:5:2)
END.

```

في بيئة تشغيل متداخل ، عادة ماتظهر بيانات المدخلات في شاشة النهاية الطرفية عند إدخالها ( أثناء تنفيذ البرنامج ) . وعلى هذا ... لاحتاج بيانات المدخلات أن تطبع مرة أخرى . وعلى أية حال ... قد لايعرف المستفيد كيف يدخل بيانات المدخلات أثناء تنفيذ البرنامج ( أى تحديد أى عناصر البيانات هو المطلوب ؟ ومتى تم إدخالها ؟ وبأى ترتيب ؟ ) . وعلى هذا ... فإن البرنامج المكتوب بصورة جيدة ليعمل في وسط تشغيل متداخل يجب أن تنتج عنه ملقنات prompts ( أى يسأل عن بيانات مدخلات ) في الأوقات المناسبة أثناء تنفيذ البرنامج . ويمكن تحقيق ذلك بإدخال عدد من عبارات write ( أو عبارات writeln ) داخل البرنامج ، كما هو موضح في المثال التالي .

مثال ( ٥ - ٤ )

دعنا نعتبر الآن صيغة لتشغيل متداخل لصيغة البرنامج الموجودة في المثال السابق . سوف ندخل بصفة خاصة مجموعة ملقنات ، بحيث يعرف المستفيد كيف يقوم بإدخال بيانات المدخلات أثناء تنفيذ البرنامج . كما أننا لن نكتب على ذلك بيانات المدخلات مع المخرجات المحسوبة ، حيث إن هذا ليس ضروريا في التشغيل المتداخل .

منطق البرنامجين الموجودين في المثالين السابقين مباشر جدا . وعلى هذا ... فلاحاجة لنا بأن نهتم بطرق بديلة لتسلسل العبارات . وعلى أية حال ... فهناك بعض المعالم المرغوب في وجودها ، والتي يمكن أن تضاف للبرنامج . فقد

```

PROGRAM pizzas(input,output);

(* This program calculates the selling price of a pizza,
   given the diameter and the number of extra items. *)

(* Interactive version *)

CONST pi = 3.14159;
      fixedcost = 0.75;
      basecost = 0.01;
      extracost = 0.0025;
VAR n : integer;
     d,area,cost,price : real;

```

```
BEGIN (* action statements *)
  page;
  write(' Enter the pizza size (diameter), in inches: ');
  readln(d);
  write(' Enter the number of extra ingredients: ');
  readln(n);
  area := pi*sqr(d)/4;
  cost := fixedcost + basecost*area + n*extracost*area;
  price := 1.5*cost;
  writeln;
  writeln(' The selling price is: $', price:5:2)
END.
```

نختار - على سبيل المثال - أن نكتب التكاليف المخططة ( أى قيم الثوابت ) مع النتائج المحسوبة . قد تكون هذه المعلومات مفيدة إذا نفذ البرنامج العديد من المرات باستخدام بيانات تكاليف مختلفة فى كل مرة من مرات التنفيذ .

وإحدى السمات التى قد تكون مفيدة ، هى إمكانية تنفيذ البرنامج على التوالى لعدة فئات مختلفة من بيانات المدخلات . وسوف نرى كيف يمكن تحقيق ذلك فى الفصل القادم .

### ٣ - إدخال البرنامج داخل الكمبيوتر : 3. ENTERING THE PROGRAM INTO THE COMPUTER

بعد كتابة البرنامج ، يجب إدخاله داخل الكمبيوتر قبل أن يمكن أن يترجم وينفذ . وعادة مايتحقق ذلك بإحدى طريقتين ممكنتين ، والطريقة الأكثر شيوعا هى استخدام منقح editor .

وعمليا تحتوى كل نظم الكمبيوتر الحديثة على بعض أنواع المنقحات التى يمكن استخدامها فى إدخال ملف نصوص text file ( ويمكن أن يكون ملف النصوص عبارة عن برنامج أو توثيق نص ، مثل الخطاب ، أو ملف البيانات وغيرها ) . وبعض المنقحات تكون موجهة للسطر line - oriented ، بينما يكون بعضها موجهة للرمز - Character orienter . والمنقحات الموجهة للرمز ، والتى تستخدم مع وحدات العرض المرئى المتداخل عادة مايشار إليها بأنها منقحات شاشة Screen editors . ومثل هذه المنقحات مريحة عمليا فى استخدامها ، حيث إنها تسمح بعرض أجزاء كبيرة من الملف على الشاشة فى أى وقت . ويمكن تحريك نقطة البداية التى تضىء ضوئا متقطعا لأى موقع فى الشاشة ، وهذا يسهل من الإدخال والتعديل والحذف فى النص .

وهناك منقحات مختلفة شائعة الاستخدام لكل أنواع الكمبيوتر . ويمكن استخدام أى منقح فى إدخال برنامج البسكال وملفات البيانات المصاحبة له ، بالرغم من أن بعض نظم البسكال تحتوى على منقحات خاصة بها . وبغض النظر عن نوع المنقح المستخدم ، فإن عملية إدخال البرنامج تكون عبارة عن كتابة برنامج البسكال داخل الكمبيوتر ، وذلك سطرًا سطرًا . ( ويمكن تحقيق ذلك باستخدام كل من المنقحات الموجهة للسطر والموجهة للرمز ) . وبعد ذلك يحدد اسم ملف البرنامج ، ويخزن البرنامج فى ذاكرة الكمبيوتر أو عن طريق إحدى وحدات التخزين المساعد . وفى بعض الأحيان تضاف لاحقة suffix ، مثل PAS إلى اسم الملف ، وذلك لتعريف الملف بأنه برنامج بسكال . ( وعادة ماتسمى مثل هذه اللاحقة بأنها اتساع extension ) . ويظل البرنامج فى هذه الصورة ، حتى يكون معدا لترجمته بمتراجم البسكال .

والطريقة البديلة لاستخدام المنقح هى قراءة برنامج البسكال فى الكمبيوتر من مجموعة من البطاقات المثقبة . وفى واقع الأمر ... فإن هذه الطريقة أصبحت متقادمة فى إدخال البرامج ، بالرغم من أنها مازالت تستخدم مع بعض أجهزة الكمبيوتر الكبيرة . واستخدام البطاقات المثقبة أقل راحة من استخدام المنقح ، إلا أنها أفضل من لاشئ .

تستخدم العديد من نظم الكمبيوتر الكبيرة كل من تشغيل الدفعة وتشغيل المشاركة الزمنية مع أحد صيغ التنقيح فى وسط الخط المفتوح . ويسمح هذا للمستخدم بأن يختار طريقة العمل الأكثر راحة له . فمثلا يمكن إدخال برنامج جديد والبيانات المصاحبة له بواسطة منقح ، إلا أن النتائج الحاسوبية ( أى محتويات ملف المخرجات ) يمكن أن تخرج عن طريق طابع أسطر .

ويجب أن يحدد القارئ نوع عملية إدخال البرنامج المتاحة له . فإذا ما كان هناك منقح متاح له ، فيجب عليه أن يحصل على دليل التشغيل ، أو على أى مجموعة تعليمات تصف له بالضبط كيفية استخدام المنقح .

#### ٤ - ترجمة وتنفيذ البرنامج : 4. COMPILING AND EXECUTING THE PROGRAM

بعد الانتهاء من إدخال البرنامج الكامل بطريقة صحيحة داخل الكمبيوتر ، يمكن ترجمته وتنفيذه . وعادة ماتتحقق عملية الترجمة تلقائيا كأستجابة لأمر معين ( مثل COMPILE ) . وفى بعض الأحيان يكون من الضروى أيضا توصيل link برنامج التشغيل المترجم إلى إجراء routines واحد أو أكثر من برامج المكتبة ، وذلك لتنفيذ البرنامج . ويتحقق ذلك تلقائيا فى معظم الحالات أيضا وذلك بإصدار أمر بسيط ( مثل LINK ) .

سوف ينتج عن الترجمة والاتصال الناجحين لبرنامج البسكال برنامج تشغيل ( بلغة الآلة ) قابل للتنفيذ ، يمكن أن ينفذ بعد ذلك إذا ما صدر أحد أوامر النظام المناسبة ( مثل EXECUTE ) . وسوف تختلف تفاصيل التنفيذ على أية حال طبقا لنوع البيئة المستخدمة فى التشغيل ... ففى بيئة غير متداخلة - على سبيل المثال - يكون مطلوبا ملف بيانات مدخلات منفصل . ويجب إدخال هذا الملف داخل الكمبيوتر قبل أن يمكن تنفيذ برنامج التشغيل . وعند تنفيذ البرنامج تقرأ بيانات المدخلات تلقائيا من ملف البيانات ، ويتم تشغيلها لينتج عنها مجموعة من بيانات المخرجات ، والتي تخزن تلقائيا فى ملف بيانات المخرجات . ويجب أن يطبع ملف المخرجات هذا ، أو يعرض على الشاشة كعملية منفصلة بعد الانتهاء من تنفيذ البرنامج .

مثال ( ٥ - ٥ )

افرض أننا نرغب فى تحديد سعر بيع الفطائر التى قطرها 10 بوصة ، بها ثلاثة مكونات إضافية . سوف نستخدم برنامج البسكال لبيئة غير متداخلة ، والموجود فى المثال رقم ٥ - ٣ .

ندخل البرنامج فى الكمبيوتر أولا ، وذلك عن طريق نهاية طرفية باستخدام منقح نصوص . ويجب أن نكون حذرين عند الإدخال ، وذلك بتصحيح كل الأخطاء . ويجب إعطاء اسم لتمييز البرنامج ، مثل PIZZAS.PAS ، بعد ذلك ندخل ملف بيانات المدخلات ، والذي يحتوى على القيمتين التاليتين :

3 10.0

وسوف يخزن هذا الملف ككيونة مستقلة تحت اسم :

INPUT.DAT.

والآن نكون مستعدين لترجمة وتوصيل البرنامج . ويمكن عمل ذلك بإدخال الأمر التالى :

COMPILE PIZZAS.PAS

عن طريق النهاية الطرفية . ( ويتم تنشيط الاتصال تلقائيا إذا ماتمت ترجمة البرنامج بنجاح ) . وعند ذلك يكون لدينا ملف جديد اسمه PIZZAS.OBJ ، والذي يحتوى على برنامج تشغيل مترجم . هذا يكون بالإضافة بالطبع إلى الملف الأساسى PIZZAS.PAS الذى يحتوى على برنامج المصدر بلغة البسكال .

ولتنفيذ برنامج التشغيل ، فإننا نكتب الأمر

EXECUTE PIZZAS

عن طريق النهاية الطرفية . ( لا تكون هناك حاجة لكتابة العنوان OBJ. ) وعلى هذا ... تنفذ التعليمات الموجودة في برنامج البسكال الأساسي ، وذلك بالرغم من أن برنامج التشغيل - وليس برنامج المصدر الأساسي - هو الذى ينفذ . وعلى هذا ... تحدث الأشياء التالية :

(١) تقرأ قيم  $n, d$  من ملف المدخلات .

(٢) تحسب المساحة والتكلفة الكلية والسعر .

(٣) نكتب قيم  $price, n, d$  في ملف المخرجات .

وهذا ينهى جزء التنفيذ للبرنامج . ولكى نرى النتائج المحسوبة على أية حال ، يكون من اللازم عرض ملف المخرجات ، وعلى هذا ... فإننا نصدر الأمر :

TYPE OUTPUT.DAT

عن طريق النهاية الطرفية . وسوف يتسبب ذلك فى عرض المخرجات التالية :

Pizza size (diameter): 10. inches  
Number of extra ingredients: 3

The selling price is: \$ 3.19

وينفذ برنامج البسكال فى التشغيل المتداخل بطريقة مختلفة . ولا يكون مطلوب ملفات بيانات مدخلات ، أو ملفات بيانات مخرجات بصفة خاصة . وبدلاً من ذلك يتم إدخال بيانات المدخلات فى الكمبيوتر أثناء تنفيذ البرنامج . وبالمثل تعرض بيانات المخرجات عند إنتاجها أثناء تنفيذ البرنامج أيضاً . وبالنسبة لمعظم التطبيقات ، فإن هذه الطريقة للتشغيل تكون أكثر راحة .

مثال (٥ - ٦)

دعنا نعيد المشكلة الموجودة فى مثال ٥ - ٥ ، باستخدام برنامج بسكال فى التشغيل المتداخل ، الموجود فى مثال ٥ - ٤ . سوف ينفذ البرنامج على جهاز ميكرو كمبيوتر ، مع استخدام نظم برامج للنظام تحتوى على منقح ، وعلى مترجم بسكال .

نكتب مرة أخرى البرنامج فى الكمبيوتر باستخدام منقح النصوص ، مع تصحيح أى أخطاء تحدث فى إدخال البرنامج . عند ذلك يخزن البرنامج كملف نصوص تحت اسم PIZZAS.PAS . لاحتاج الآن على أية حال ملفاً للمدخلات ، وبدلاً من ذلك ... فإننا نستمر فى ترجمة وتوصيل البرنامج بإدخال الأوامر :

COMPILE PIZZAS.PAS

LINK PIZZAS

فإننا مانفذ الأوامر LINK , COMPILE بنجاح ؛ فسوف ينتج عن ذلك برنامج تشغيل (PIZZAS.OBJ) ، وبعد ذلك نستطيع أن نستمر فى التنفيذ .. وإلا فإنه يكون من الضروري العودة إلى برنامج المصدر (PIZZAS.PAS) ، وإجراء أى تعديلات ضرورية ( عن طريق المنقح ) ثم نبدأ العملية بنفس التسلسل مرة أخرى . ( هذه العملية المتكررة يجب أن تتبع بالطبع فى كل من صيغ بسكال المتداخلة وغير المتداخلة ) .

دعنا نفترض الآن أن الترجمة والتوصيل حدثا بنجاح . ولتنفيذ البرنامج ، فإننا نكتب :

EXECUTE PIZZAS

ويبدأ عند ذلك تنفيذ التعليمات الموجودة في برنامج البسكال الأصلي . وسوف يلقي الكمبيوتر بصفة خاصة الحاجة إلى حجم القطيرة ، وذلك بإنتاج الرسالة التالية على الشاشة :

Enter the pizza size (diameter), in inches:

يستجيب المستخدم على ذلك بإدخال القيمة 10.0 ، ويسأل الكمبيوتر بعد ذلك عن عدد العناصر الإضافية بإنتاجه الرسالة التالية :

Enter the number of extra ingredients:

ويدخل المستخدم عند ذلك القيمة 3 .

وكل بيانات المدخلات المطلوبة تم إدخالها الآن . وعلى هذا ... تحسب المساحة والتكلفة الكلية والسعر مع عرض السعر على الشاشة في الصورة التالية :

The selling price is: \$ 3.19

لاحظ أن النتيجة النهائية ظهرت مباشرة كما حسبت ، فليس هناك حاجة لملف بيانات المخرجات .

وفيما يلي ملخصاً لجزء التداخل ، مع وضع خط تحت مايرد به المستخدم ، وذلك بغرض التوضيح :

(program statements . . . entered via an editor)

COMPILE PIZZAS.PAS

LINK PIZZAS

EXECUTE PIZZAS

Enter the pizza size (diameter), in inches: 10.0

Enter the number of extra ingredients: 3

The selling price is: \$ 3.19

الاجراءات الموضحة في المثالين السابقين تمثل الإجراءات شائعة الاستخدام ، وذلك بالرغم من إمكانية وجود بعض الاختلافات من جهاز الكمبيوتر لجهاز آخر . فيوجد على سبيل المثال بعض نظم بسكال تستخدم أمراً معيناً ( مثل RUN ) يتسبب في ترجمة برنامج المصدر المكتوب بالبسكال ، مع توصيله وتنفيذه على التوالي . ويجب على القارئ أن يحدد الأوامر الخاصة بالترجمة والتوصيل والتنفيذ المتاحة مع الجهاز المتاح له .

وفي النهاية يجب أن يتذكر القارئ أن الترجمة والتوصيل والتنفيذ الناجحة لبرنامج البسكال عادة ماتتطلب محاولات متعددة بسبب الأخطاء التي تحدث في معظم البرامج الجديدة . وهذا صحيح بالنسبة للبرامج التي يكتبها كل من المبرمجين ذوي الخبرة والمبتدئين . وعلى هذا ... فقد تبدو الإجراءات مرهقة ، وذلك بالنسبة للمبتدئين بصفة خاصة . ويجب على أية حال فهم أن تكرار حدوث مثل هذه الأخطاء سوف يقل مع اكتساب المبرمج الخبرة . وعلى هذا ... فيجب أن يكون المبتدئون حذرين من ناحية أنه قد يحدث لهم إحباط مع أول عدة برامج ، في نفس الوقت الذي تتحسن فيه قدرتهم .

## 5. ERROR DIAGNOSTICS

## ٥ - تشخيص الخطأ :

عادة ماتظل أخطاء البرمجة غير مكتشفة ، حتى تحدث محاولة ترجمة البرنامج . وبمجرد إصدار الترجمة COMPILE ، تظهر الأخطاء ، حيث إنها تمنع من ترجمة البرنامج بنجاح . وبعض هذه الأخطاء شائعة الاستخدام عادة ماتوضح على أنها ثوابت ومتغيرات غير صحيحة ، أو إشارة إلى متغير غير موضح أو استخدام تنقيط خاطئ . يشار إلى مثل هذه الأخطاء بأنها أخطاء تكوينية syntactical errors ، أو أخطاء فى القواعد grammatical errors .

وسوف تنتج معظم صيغ البسكال رسائل تشخيصية عند اكتشاف الأخطاء التكوينية . وعادة مالاتكون هذه الأخطاء التكوينية مباشرة فى معناها ، كما أنها لاتساعد على تعريف طبيعة وموقع الخطأ .

## مثال (٥-٧)

فيما يلى صيغة متداخلة أخرى من صيغ برامج البسكال الموجود فى مثال ٥ - ٤ ، والذي يحسب سعر بيع الفطيرة إذا ما أعطى حجم الفطيرة ( القطر ) ، وعدد المكونات الإضافية . ويختلف هذا البرنامج عن الصيغة السابقة فى أنه يحتوى على العديد من الأخطاء التكوينية .

وعند محاولة ترجمة البرنامج نحصل على قائمة الأخطاء التالية :

```
PROGRAM pizzas(input,output);

(* This program calculates the selling price of a pizza,
   given the diameter and the number of extra items.

(* Interactive version *)

CONST pi = 3.14159
      fixedcost = 0.75;

***** ERROR 346 MISSING ;
      basecost := 0.01;

***** WARNING 149 := ASSUMED =
      extracost = 0.0025;
VAR n : integer;
    d,area,cost,price = real;

***** WARNING 148 = ASSUMED :

BEGIN (* action statements *)
    page;
    write(' Enter the pizza size (diameter), in inches: ');

***** WARNING 105 END OF STRING NOT FOUND

***** WARNING 169 INSERT )
    readln(d);

***** ERROR 185 INVALID SYMBOL BEGIN SKIP

***** WARNING 164 INSERT ;

***** ERROR 186 END SKIP
```

(تكملة البرنامج فى الصفحة التالية)

```

***** ERROR 185 INVALID SYMBOL BEGIN SKIP
^
***** ERROR 186 END SKIP
      write(' Enter the number of extra ingredients: ');
      readln(n);
      area = pi*sqr(d)/4;
      ^
***** WARNING 150 = ASSUMED :=
      cost := fixedcost + basecost*area + n*extracost*area
      ^
***** WARNING 164 INSERT ;
      price := 1.5cost;
      ^
***** ERROR 185 INVALID SYMBOL BEGIN SKIP
^
***** ERROR 186 END SKIP
      writeln;
      writeln(' The selling price is: $', price,5,2)
      END
***** ERROR 146 UNEXPECTED END OF FILE
***** FATAL PROGRAM ERRORS - COMPILER CANNOT CONTINUE!

```

ويجب ذكر عدة نقاط . أولا : لاحظ أنه عادة ماتكون رسائل الخطأ خفية ، وبعضها يكون من الصعب فهمه . ثانيا : لاحظ أن العديد من هذه الأخطاء يتم تصحيحها تلقائيا . وتعرف هذه التصحيحات بأنها رسائل تحذيرية WARNING ، بدلا من أنها رسائل خطأ ERROR . وأخيرا لاحظ أن عدم وجود ( \* فى نهاية أول تعليق لم تكتشف X ) ( وقد تم تفسير التعليقين كتعليق واحد طويل يحتوى على أقواس داخلية ) .

وسوف تتغير رسائل الخطأ والرسائل التحذيرية من صيغة لأخرى من صيغ البسكال . وهذه الأخطاء تمثل الأخطاء المعتادة فى برامج البسكال .

وعادة ماتكون الأخطاء التكوينية واضحة جدا ، نظرا للأعراض التى تنتجها . أما الأخطاء المنطقية ، فتكون غير واضحة . وهنا يحمل البرنامج تعليمات البرنامج بطريقة صحيحة ، وهى خالية من الأخطاء التكوينية ، إلا أن هناك تعليمات خاطئة منطقيا .

وفى بعض الأحيان ينتج عن الخطأ المنطقى شرط يمكن للكمبيوتر أن يميزه . وقد ينتج مثل هذا الموقف من ظهور قيم عددية كبيرة ( تتعدى أقصى عدد مسموح بتخزينه فى الكمبيوتر ) ، أو من محاولة حساب الجذر التربيعى لعدد سالب ، وما إلى ذلك من أخطاء . وعادة ماتظهر رسائل تشخيصية فى مثل هذه المواقف لتسهيل تعريف الأخطاء وتصحيحها . وعادة ماتسمى هذه التشخيصيات بأنها تشخيصيات التنفيذ EXECUTION ، وذلك لتمييزها عن تشخيصيات الترجمة COMPILATION التى سبق ذكرها .

### مثال ( ٥ - ٨ )

الجذور الحقيقية لمعادلة من الدرجة الثانية : افرض أنه مطلوب حساب الجذور الحقيقية للمعادلة من الدرجة الثانية التالية :

$$ax^2 + bx + c = 0$$



باستخدام الصيغة التالية :

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

وفيما يلي برنامج بسكال يجرى هذه الحسابات :

```
PROGRAM realroots(input,output);

(* This program calculates the real roots of a quadratic equation *)

VAR a,b,c,d,x1,x2 : real;

BEGIN (* action statements *)
  readln(a,b,c);
  d := sqr(b)-4*a*c;
  x1 := (-b+sqr(d))/(2*a);
  x2 := (-b-sqr(d))/(2*a);
  writeln(' a=',a, ' b=',b, ' c=',c);
  writeln;
  writeln(' x1=',x1, ' x2=',x2)
END.
```

هذا البرنامج لا يحتوى على أى أخطاء تكوينية على الإطلاق ، إلا أنه لا يستطيع أن يؤدي الحسابات لقيم سالبة لـ :

$$\text{sqr}(b) - 4 \cdot a \cdot c.$$

والأكثر من هذا ... قد تظهر صعوبات عديدة إذا ما كانت هناك قيمة صغيرة جداً أو قيمة كبيرة جداً للمتغير ، أو إذا ما كانت  $a = 0$  . وفى كل من هذه الحالات يظهر تشخيص تنفيذ .

افرض على سبيل المثال أن المدخلات التالية استخدمت عند تشغيل البرنامج :

$$a=1.0 \quad b=2.0 \quad c=3.0$$

تحدث الترجمة دون أى صعوبة . وعند تنفيذ برنامج التشغيل تظهر رسالة الخطأ التالية :

```
? Error: Sqrt of Negative Argument
Error Code 2104
```

وعند ذلك ينتهى كل شئ ، حيث إن تنفيذ البرنامج لا يمكن أن يستمر بعد الوصول الى هذه النقطة .

وبالمثل افرض أن المدخلات التالية استخدمت عند تشغيل البرنامج :

$$a = 1E-30 \quad b = 1E+10 \quad c = 1E+36 \quad \text{6}$$

عند ذلك ينتج النظام رسالة الخطأ التالية :

```
? Error: REAL Math Overflow
Error Code 2101
```

وذلك عند محاولة تنفيذ البرنامج .

## 6. LOGICAL DEBUGGING

### ٦ - تصحيح المنطق :

سبق أن رأينا أن الأخطاء التكوينية ، وبعض الأخطاء المنطقية تتسبب في إظهار رسائل تشخيصية عند ترجمة البرنامج أو تنفيذه . ومن السهل اكتشاف مثل هذه الأخطاء وتصحيحها ، إلا أن بعض أنواع الأخطاء المنطقية يكون من الصعب جدا اكتشافها ، حيث إنه في مثل هذه الحالات تبدو مخرجات البرنامج كما لو كانت خالية من الأخطاء المنطقية ، حتى عند معرفة أنها موجودة ( كما في حالة ظهور مخرجات تبدو أنها صحيحة ) . وعلى هذا ... تكون هناك حاجة إلى اكتشاف مثل هذه الأخطاء وتصحيحها . والعمل الذي يؤدي لتحقيق هذا الغرض يسمى بتصحيح المنطق .

### Detecting Errors

#### اكتشاف الأخطاء :

إن أول خطوة في مهاجمة الأخطاء المنطقية هو تحديد مواقع تواجدها . ويمكن تحقيق ذلك في بعض الأحيان باختبار البرنامج الجديد ببيانات اختبارية ، فإذا لم يتم الحصول على نفس النتائج المرجوة ، فهذا يؤكد أن البرنامج به أخطاء . حتى إذا ماتم الحصول على نتائج صحيحة ، فلانستطيع التأكد تماما بأن البرنامج خال من الأخطاء ، حيث إن بعض الأخطاء تتسبب في ظهور نتائج خاطئة تحت ظروف معينة فقط . ( كما في حالة قيم معينة لبيانات المدخلات ، أو في حالة تنفيذ اختيارات معينة للبرنامج ) . وعلى هذا ... يجب أن يمر البرنامج الجديد خلال اختبار دقيق قبل أن يعتبر صحيحا . وهذا صحيح بصفة خاصة بالنسبة للبرامج المعقدة ، والبرامج التي سوف يستخدمها العديد من الناس .

وكقاعدة عامة ... تجرى الحسابات يدويا بمساعدة حاسبة جيب ، بهدف معرفة الإجابة . وبالنسبة لبعض المشاكل - على أية حال - فإن كمية العمل اللازمة لأداء مثل هذه الحسابات تكون كبيرة جدا ، وتمنع من إجراء مثل هذا الاختبار . ( تذكر أن الحسابات التي تستغرق عدة دقائق لتنفيذها بواسطة الكمبيوتر قد تتطلب عدة أسابيع لحلها يدويا ) . وعلى هذا ... فإن حساب العينة لا يمكن استخدامه بصفة دائمة لاختبار البرنامج الجديد . وتصحيح منطق مثل هذه البرامج يمكن أن يكون صعبا بصفة خاصة ، بالرغم من أن المبرمج عادة ما يمكنه اكتشاف وجود أخطاء منطقية عن طريق دراسة النتائج التي يتم الحصول عليها من البرنامج دراسة دقيقة لمعرفة ما إذا كانت مناسبة أم لا .

### Correcting Errors

#### تصحيح الأخطاء :

بعد معرفة أن البرنامج يحتوي على أخطاء منطقية ، تكون هناك حاجة إلى إيجاد هذه الأخطاء . واكتشاف الأخطاء عادة ما يبدأ بمراجعة دقيقة لكل مجموعة من مجموعات العبارات المنطقية داخل البرنامج . وبمعرفة أنه هناك خطأ معين في مكان ما ، فعادة ما يستطيع المبرمج أن يحدد موقع الخطأ عن طريق هذه الدراسة الدقيقة . فإذا لم يستطع أن يجد الخطأ ، فترك البرنامج لفترة معينة يساعد في بعض الأحيان ( وهذا صحيح بصفة خاصة إذا ما كان المبرمج مرهقا من العمل ) . وليس من غير المعتاد للمبرمج أن يمر على الخطأ دون اكتشافه من أول مرة .

وإذا لم يمكن تحديد موقع الخطأ عن طريق فحص البرنامج ، فيجب تعديل البرنامج لطبيع نتائج مرحلية معينة ، ثم يستمر في التنفيذ . ( مثل هذا الأسلوب يشار إليه في بعض الأحيان بأنه تتبع TRACING ) . وعادة ما يصبح مصدر الخطأ واضحا عند فحص هذه النتائج المرحلية فحصا دقيقا . وعادة ما يستطيع المبرمج تعريف إحدى مناطق البرنامج والتي يحدث داخلها الخطأ . وكلما ازداد حجم المخرجات المرحلية ، ازدادت فرصة تحديد مصدر الخطأ .

وفي بعض الأحيان لا يمكن تحديد موقع الخطأ ببساطة ، بالرغم من استخدام معظم أساليب التصحيح . وفي مثل هذه المواقف يتوقع للمبرمج المبتدئ أنه يواجه مشكلة خارج حدود معرفته ، مثل أخطاء نظم المكونات ، أو أخطاء المترجم نفسه . وفي كل الحالات تقريبا تكون المشكلة في منطق البرنامج . وعلى هذا ... يجب على المبرمج المبتدئ أن يقاوم إغراء إلقاء اللوم على الكمبيوتر ، ويفحص برنامجا بدقة ليكتشف أخطاء البرمجة . ( أخطاء نظم المكونات نادرة الحدوث ، كما أن أخطاء المترجم لا تظهر عادة إلا عند استخدام مترجم جديد ، لكن يتم تصحيحها خلال فترة بسيطة من استخدام المترجم ) .

وأخيرا يجب أن يميز القارئ حقيقة أن بعض الأخطاء المنطقية غير قابلة للهروب في برمجة الكمبيوتر . وعلى هذا ... يجب على المبرمج الواعي أن يبذل كل المحاولات لتقليل حدوث هذه الأخطاء . وعلى هذا ... يجب على المبرمج أن يتوقع أن هناك حاجة إلى تصحيح منطقي عند كتابته برامج بسكال واقعية وذات معنى .

### مثال (٥-٩)

تقديم كثيرة حدود : لقد كتب أحد الطلبة برنامج بسكال لتقويم كثيرة حدود لها الشكل التالي :

$$y = \left(\frac{x-1}{x}\right) + \frac{1}{2}\left(\frac{x-1}{x}\right)^2 + \frac{1}{3}\left(\frac{x-1}{x}\right)^3 + \frac{1}{4}\left(\frac{x-1}{x}\right)^4 + \frac{1}{5}\left(\frac{x-1}{x}\right)^5$$

والكى يبسط الطالب البرمجة ، فقد قام بتعريف متغير جديد  $u$  على أنه :

$$u = \left(\frac{x-1}{x}\right)$$

وبذلك تحولت الصيغة إلى مايلي :

$$y = u + \frac{u^2}{2} + \frac{u^3}{3} + \frac{u^4}{4} + \frac{u^5}{5}$$

وفيما يلي البرنامج الكامل الذى أعده الطالب :

```
PROGRAM formula(input,output);

(* Program to evaluate an algebraic formula *)

VAR u,x,y : real;
BEGIN
  readln(x);
  u := x-1/x;
  y := u+sqr(u/2)+(u/3)*sqr(u/3)+sqr(u/4)*sqr(u/4)+(u/5)*sqr(u/5)*sqr(u/5);
  writeln(' x=',x,'      y=',y)
END.
```

ويعرف الطالب أن قيمة  $y$  يجب أن تكون حوالى 0.69 عندما تكون  $x = 2$  ، إلا أن مخرجات البرنامج كانت على النحو التالى :

x= 2.0000000E+00      y= 2.2097050E+00

وقد استنتج الطالب - على ذلك - أن البرنامج يحتوى على أخطاء منطقية ، والتي يجب أن يعرفها ويصححها .

وبعد الفحص الدقيق للبرنامج تحقق الطالب من أن أول عبارة تحديد غير صحيحة . ويجب أن تكون على الصورة التالية :

$$u := (x-1)/x;$$

وقد قام الطالب بتصحيح البرنامج ، وأعاد تنفيذه ، مستخدماً  $x = 2$  ؛ وحصل على المخرجات التالية :

x= 2.0000000E+00      y= 5.6738380E-01

وهذا يبين أنه مازال هناك خطأ منطقي .

وبعد دراسة إضافية أخرى للبرنامج ، اكتشف أن عبارة التحديد الثانية خاطئة أيضا . ويجب أن تكون على الصورة التالية :

$$y := u + \text{sqr}(u) / 2 + (u + \text{sqr}(u)) / 3 + (\text{sqr}(u) * \text{sqr}(u)) / 4 + (u + \text{sqr}(u) * \text{sqr}(u)) / 5;$$

فقام بتعديل البرنامج وإعادة تنفيذه ، ليحصل على النتيجة الصحيحة التالية :

$$x = 2.0000000E+00 \quad y = 6.8854160E-01$$

## Review Questions

## أسئلة للمراجعة :

- (١) ماذا تعنى البرمجة من القمة إلى القاعدة ؟ وماهى مميزاتها ؟ وكيف يمكن تنفيذها ؟
- (٢) ماهى الشفرة الشبيهة ؟
- (٣) ماذا تعنى البرمجة من القاعدة إلى القمة ؟ وماهو الفرق بينها وبين البرمجة من القمة إلى القاعدة ؟
- (٤) لخص التكوين الشامل لبرنامج البسكال .
- (٥) كيف تستخدم الفاصلة المنقوطة فى البسكال ؟ وفى أى مكان يجب ظهورها ؟
- (٦) ماهى علامة التنقيط الخاصة التى يجب أن تظهر فى نهاية كل برنامج من برامج البسكال ؟
- (٧) ماهو الغرض من BEGIN , END ؟ وماهو التنقيط الذى يصاحب هاتين الكلمتين الأساسيتين ؟
- (٨) ماهى درجة المرونة المتاحة للمبرمج فى تسلسل العبارات المنطقى داخل برنامج البسكال ؟ وضح إجابتك .
- (٩) لماذا ترحل بعض العبارات داخل برنامج البسكال ؟ وهل هذا الترحيل ضرورى ؟
- (١٠) ماهى أسباب وضع التعليقات فى برنامج البسكال ؟ وماهى الكثافة التى يجب أن تظهر بها هذه التعليقات ؟
- (١١) اذكر عاملين يسهمان فى إنتاج بيانات مخرجات واضحة .
- (١٢) وضح كيف تؤثر بيئة البرمجة ( بيئة متداخلة وبيئة غير متداخلة ) على المعالم الخاصة الموجودة فى البرنامج للمدخلات والمخرجات .
- (١٣) ماهو التلقين ؟ تحت أى ظروف يجب أن يظهر التلقين فى برنامج البسكال ؟
- (١٤) ماهو ملف النصوص ؟
- (١٥) ماهو الفرق بين المنقح الموجه للسطر ، والمنقح الموجه للرمز ؟
- (١٦) اذكر طريقتين مختلفتين لإدخال برنامج البسكال فى الكمبيوتر . أى هذه الطرق تفضلها ؟
- (١٧) ماهو الفرق بين ترجمة برنامج البسكال وتنفيذه ؟
- (١٨) مامعنى ربط linking ؟ وكيف يختلف الربط عن الترجمة ؟

(١٩) اذكر بعض الاختلافات الأساسية في طرق تنفيذ برنامج البسكال طبقاً لما إذا كان وسط الحسابات متداخلاً أم لا .

(٢٠) ماذا يعنى الخطأ التكويني ؟

(٢١) ماذا يعنى الخطأ المنطقي ؟ كيف تختلف الأخطاء التكوينية عن الأخطاء المنطقية ؟

(٢٢) اذكر بعض الأخطاء التكوينية الشائعة .

(٢٣) اذكر بعض الأخطاء المنطقية الشائعة .

(٢٤) ماهي الرسائل التشخيصية ؟

(٢٥) ماهو الفرق بين تشخيصات الترجمة وتشخيصات التنفيذ ؟ اذكر بعض المواقف التي تظهر فيها رسائل تشخيصية .

(٢٦) مامعنى تصحيح المنطق ؟ اذكر بعض إجراءات التصحيح الشائعة .

(٢٧) مامعنى التتبع tracing ؟ وكيف يكون مفيداً ؟

## Problems

## مشاكل :

التمارين التالية تهتم بجمع المعلومات أكثر من اهتمامها بحل المشاكل .

(٢٨) إذا ماكانت المشاركة الزمنية مستخدمة في كليتك أو في مكتبك ، فاحصل على إجابة للأسئلة التالية :

أ ( هل توجد نهايات طرفية للعرض المرئي ؟ هل توجد نهايات طرفية لطباعة نسخ دائمة ؟

ب ( كيف يمكن بدء وإنهاء عمل النهاية الطرفية ؟

ج ( كيف يمكن حذف رمز من رموز أحد الأسطر المكتوبة قبل إرسال السطر إلى الكمبيوتر ؟ وكيف يمكن حذف سطر كامل ؟

د ( كيف يمكن نقل السطر إلى الكمبيوتر ؟

هـ ( هل يمكنك الحصول على نسخة دائمة ؟ إذا كانت الإجابة نعم ، فحدد كيف يتم ذلك .

و ( هل من المطلوب إجراء اتصال هاتفي لعمل اتصال مع الكمبيوتر ؟ إذا كانت الإجابة نعم ، فما هي الإجراءات التي تتبع ؟

ز ( حدد بالضبط كيف تبدأ العمل مع الكمبيوتر الكبير وكيف تنتهي .

ح ( هل يمكن للنهاية الطرفية أن تعمل بطريقة محلية ( أى تعمل كوحدة مستقلة بذاتها منفصلة عن عمل الكمبيوتر الكبير ) ؟

ط ( ماهو المنقح أو المنقحات المتاحة في النظام ؟ وكيف تجرى وظائف التنقيح المعتادة ( مثل الحذف والإضافة وغيرها ) ؟

ى ( كيف يمكن الاتصال بمترجم البسكال فى النظام المتاح لك ؟ وماهى الإجراءات اللازمة لترجمة البرنامج ولتوصيله وتنفيذه ؟

ك ( ماهى تكلفة استخدام الكمبيوتر ؟

(٢٩) إذا ما كان تشغيل الدفعة مستخدما فى كليتك أو فى مكتبك ، فاحصل على إجابة للأسئلة التالية :

أ ( هل توجد آلات تثقيب بطاقات ؟ إذا كانت الإجابة نعم ، فكيف يتم فتحها وإغلاقها ؟ كيف تتم تغذية البطاقات داخل الآلة ؟ كيف تثقب البطاقة ؟ كيف يعاد إنتاج بطاقة أخرى ؟

ب ( ماهو موقع قارئ البطاقات ؟ كيف يمكن قراءة مجموعة من البطاقات المثقبة بواسطة قارئ البطاقات ؟

ج ( ماهى بطاقات التحكم المطلوبة ؟

د ( كيف يمكن الاتصال بمترجم البسكال فى النظام المتاح لك ؟ وماهى الإجراءات اللازمة لترجمة البرنامج ولتوصيله وتنفيذه ؟

هـ ( كيف يمكن تخزين البرنامج أو ملف البيانات فى هذا النظام ؟ وكيف يمكن الاتصال بالبرنامج أو بملف البيانات ؟ وكيف يمكن حذف أى منهما ؟

و ( حدد موقع طابع الأسطر . ماهو شكل المخرجات المطبوعة ؟

ز ( ماهى تكلفة استخدام الكمبيوتر ؟

(٣٠) إذا ما كان يستخدم جهاز ميكروكمبيوتر فى كليتك أو فى مكتبك ، فاحصل على إجابة للأسئلة التالية :

أ ( حدد بالضبط المعدات المتاحة لك ( طابعات ، أو وحدات ذاكرة مساعدة وغيرها ) .

ب ( كيف يمكن فتح وإغلاق الجهاز ؟

ج ( كيف تخزن البرامج ؟ وكيف تعرض ؟ وكيف تنتقل من إحدى أنواع الذاكرات إلى ذاكرة أخرى ؟

د ( كيف يمكن حذف رمز واحد من رموز أحد الأسطر المكتوبة قبل إرسال السطر إلى الكمبيوتر ؟ وكيف يمكن حذف سطر كامل ؟ .

هـ ( كيف يمكن نقل السطر إلى الكمبيوتر ؟

و ( كيف يمكن الاتصال بالمنقح ؟ وكيف يمكن أداء وظائف التنقيح المعتادة ( مثل الحذف والإضافة وغيرها ) ؟

ز ( كيف يمكن الاتصال بمترجم البسكال فى النظام المتاح لك ؟ وماهى الإجراءات اللازمة لترجمة البرنامج ولتوصيله وتنفيذه ؟

ح ( ماهى تكلفة استخدام الكمبيوتر ؟

## Programming Problems

مشاكل برمجة :

(٢١) يمثل مثال ١ - ٧ فى الفصل الأول من الكتاب برنامجا لحساب مساحة الدائرة بمعرفة نصف قطر الدائرة . قم بإدخال هذا البرنامج فى الكمبيوتر المتاح لك . تأكد من تصحيح أى أخطاء تحدث عند كتابة البرنامج . اسرد محتويات البرنامج بعد تخزينه فى الكمبيوتر . عندما تتأكد من صحة البرنامج ، ترجم البرنامج ، ثم نفذ برنامج التشغيل ، مستخدما قيما مختلفة عديدة لنصف القطر . تحقق من صحة الإجابات المحسوبة ، وذلك بمقارنتها بحسابات تعدها يدويا .

(٢٢) أعد حل المشكلة السابقة بالنسبة لبرنامج واحد أو أكثر من البرامج المعطاه فى المشكلة رقم ١ فى الفصل الأول من الكتاب .

(٢٣) يقدم مثالى ٥ - ٣ و ٥ - ٤ صيغتين مختلفتين لبرنامج بسكال يحدد سعر بيع الفطائر عند معرفة حجم الفطائر وعدد المحتويات الإضافية . اختر الصيغة التى تناسب وسط التشغيل المتاح لك . أدخل البرنامج فى الكمبيوتر ، وخرنه ، ثم قم بتنفيذ البرنامج ، مستخدما مجموعات مختلفة من بيانات المدخلات . استخدم النتائج المحسوبة فى إعداد قائمة مثل التى تتوقع أن تراها فى محل الفطائر ، موضحا أسعار كل نوع من أنواع الفطائر بالنسبة للحجم ، وعدد المحتويات الإضافية .

(٢٤) اكتب برنامج بسكال كاملا لكل موقف من المواقف التالية . ادخل كل برنامج فى الكمبيوتر بعد التأكد من صحة أخطاء كتابة البرنامج . خزن البرنامج ، ثم قم بطباعة قائمة به . عندما تتأكد من أن البرنامج قد تم إدخاله بطريقة صحيحة ، قم بترجمة البرنامج وتنفيذه . أعد تكرار ذلك كلما دعت الحاجة لذلك ، حتى تحصل على برنامج خال من الأخطاء . استخدم أسلوب البرمجة التقليدى كلما كان ذلك ممكنا .

١ ( اطبع HELLO فى أول السطر .

ب ( اجعل الكمبيوتر يطبع .

HI, WHAT'S YOUR NAME?

فى سطر واحد . عند ذلك يقوم المستفيد بإدخال اسمه بعد علامة الاستفهام مباشرة . عند ذلك يترك الكمبيوتر سطرين فارغين ، ويطبع :

WELCOME (name)

LET'S BE FRIENDS!

على سطرين متتاليين . افرض أن اسم المستفيد يحتوى على 6 خانات بالضبط ( أضف فراغات فى نهاية الاسم إذا كانت هناك حاجة لذلك ) .

ج ( حول قراءات درجة الحرارة من درجات فهرنهايت إلى درجات مئوية ، مستخدما العلاقة التالية :

$$C = (5/9)*(F-32)$$

اختبر البرنامج ، مستخدما القيم التالية -22 , -200 , 68 , 212 , 150 درجة فهرنهايت .

د ( حدد كمية النقود الموجودة فى خزانة أحد البنوك تحتوى على  $n_1$  من الريال الفضة ، و  $n_2$  من نصف ريال ، و  $n_3$  من ربع ريال ، و  $n_4$  من عشرة هللة ، و  $n_5$  من خمسة هللة . استخدم القيم التالية لاختبار البرنامج :  $n_3 = 3$  ,  $n_2 = 7$  ,  $n_1 = 11$  ,  $n_5 = 17$  ,  $n_4 = 12$  .

هـ) احسب حجم الكرة ومساحتها ، مستخدماً الصيغتين التاليتين :

$$V = \frac{4\pi r^3}{3}$$

$$A = 4\pi r^2$$

اختبر البرنامج ، مستخدماً القيم التالية للقطر : 0.2 و 6 و 12.2 .

و) احسب كتلة الهواء الموجودة في عجلة سيارة ، مستخدماً العلاقة التالية :

$$PV = 0.37m(T + 460)$$

حيث  $P$  = الضغط بالرطل على البوصة المربعة .

$V$  = الحجم بالقدم المكعب .

$m$  = كتلة الهواء بالرطل .

$T$  = درجة الحرارة بالدرجات الفهرنهايتية .

وتحتوى العجلة على 2 قدم مكعب من الهواء . افرض أن الضغط هو 32 رطلاً على البوصة المربعة في درجة حرارة الغرفة .

ز) اقرأ كلمة مكونة من خمسة حروف في الكمبيوتر ، ثم قم بعمل شفرة الكلمة حرفاً حرفاً ، وذلك بطرح 30 من القيمة العددية المستخدمة في تمثيل كل حرف . وعلى هذا ... فإذا كانت شفرة ASCII مستخدمة في تمثيل الرموز ، فإن الحرف a (والذى يمثل بالعدد 97) سوف يصبح C (والممثل بالعدد 67) وهكذا .

اكتب صيغة شفرة الكلمة كمخرجات . اختبر البرنامج مستخدماً الكلمات التالية : Japan , roses , white , zebra . ملاحظة : لاحظ أن  $C = \text{chr}(\text{ord}(a) - 30)$  .

ح) اقرأ الكلمة المكونة من خمسة حروف في الكمبيوتر ، والتي سبق عمل تخطيط شفرة لها في الحالة السابقة . قم بعمل شفرة للكلمة بإعادة الخطوات السابقة بصورة عكسية ، ثم قم بطباعة الكلمة الجديدة .



## الفصل السادس

### مكونات التحكم

## Control Structures

فى كل البرامج التى تعرضنا لها حتى الآن كانت تنفذ كل التعليمات مرة واحدة بنفس ترتيب ظهورها فى البرنامج . ومثل هذه البرامج تكون بسيطة وغير واقعية ، حيث إنها لاتحتوى على أى منطق لمكونات التحكم مثل اختبارات تحديد ما إذا كانت إحدى الشروط صحيحة أو خاطئة أو تكرار تنفيذ مجموعة من العبارات أو اختيار إحدى مجموعات العبارات من عدة إمكانيات مختلفة ، إلا أن معظم البرامج العملية تستخدم مثل هذه المعالم استخداما كثيرا . وعلى هذا ... يجب أن نتعلم كيفية استخدام منطق مكونات التحكم فى برامجنا ، حتى يمكن أن نأخذ فى الاعتبار مواقف مشاكل أكثر واقعية وأكثر أهمية .

فمثلا ، تتطلب العديد من البرامج تكرار تنفيذ مجموعة من التعليمات المتتالية ، حتى يتحقق أحد الشروط المنطقية . وبصفة عامة ... فإن عدد التكرارات المطلوب لا يكون معروفا مسبقا . ومثل هذا التكرار يعرف بأنه دورة شرطية conditional looping . كما أنه هناك عملية أخرى هى الدورة غير الشرطية unconditional looping ( أو الدورة looping المسطحة ) والتى يحدث فيها تكرار مجموعة من التعليمات المتتالية عدد محدد من المرات . ويوجد موقف آخر يتكرر ظهوره ، وهو الحاجة إلى إجراء اختبار منطقي ، ثم أخذ بعض الإجراءات الخاصة التى تعتمد على ناتج هذا الاختبار . وهذا ما يعرف بالتنفيذ الشرطى conditional execution . وأخيرا هناك نوع خاص من التنفيذ الشرطى ، حيث يتم فيه اختيار مجموعة معينة من العبارات ، وذلك من العديد من المجموعات المتاحة . ويشار فى بعض الأحيان لذلك بالاختيار selection .

ويمكن أداء كل هذه العمليات بسهولة فى البسكال . وسوف نرى كيف يتحقق ذلك فى الفصل الحالى . واستخدام هذه المادة يفتح الباب لمشاكل برمجة أوسع كثيرا وأكثر أهمية .

### 1. PRELIMINARIES

#### ١ - مبادئ :

قبل اعتبار تفاصيل مكونات التحكم فى البسكال ، دعنا نراجع مفاهيم معينة سبق تقديمها فى الفصل الثانى والفصل الثالث . ويجب استخدام هذه المفاهيم مع مكونات التحكم . وعلى هذا ... فإن فهمهم ضرورى قبل الاستمرار فى المزيد .

تذكر أولا أن تعبير البوليان يمثل شرطا وهذا الشرط إما أن يكون صحيحا أو خاطئا ( انظر قسم ١٠ من الفصل الثانى ) . وتتكون تعبيرات بوليان من مجموعة من العناصر من نفس النوع ( أى نوع باستثناء النوع البوليانى ) مع واحد أو أكثر من المؤثرات العلاقية relational operators السبعة هى : < , <= , > , >= , IN ( انظر قسم ٤ من الفصل الثالث ) . وحتى الآن ناقشنا أول ستة مؤثرات علاقية فقط ، أما المؤثر السابع ( IN ) ، فيختلف عنها ، ولن نتعرض له حتى الفصل الثانى عشر من الكتاب .

#### مثال (٦-١)

فيما يلى العديد من تعبيرات بوليان :

(انظر الصفحة القادمة)

```
count <= 100

sqrt(a+b+c) > 0.005

answer = 0

balance >= cutoff

ch1 < 'T'.
```

تحتوى أول أربعة تعبيرات على عناصر عددية ، وفى الواقع يجب أن تكون قراءتها واضحة .

يفترض فى التعبير الأخير أن ch1 متغير حرفى . ويكون هذا التعبير صحيحا إذا ما أتت الرموز الممثلة بواسطة ch1 قبل T فى فئة الرموز ، إلا إذا ما كانت ord(ch1) < ord('T') ، وإلا فإن التعبير يكون خاطئا .

ويحتوى البسكال بالإضافة إلى المؤثرات العلاقية على مؤثرات منطقية - logical oper ( NOT , AND , OR ) tors تستخدم مع عناصر بوليان ( قسم ٤ من الفصل الثالث ) . أول مؤثرين (AND , OR) يستخدمان مع عناصر بوليان لتكوين تعبيرات منطقية . أما الثالث (NOT) ، فهو بادئة تستخدم فى نفي عنصر بوليان .

## مثال (٦-٢)

فيما يلى تعبيرات بوليان توضح استخدام المؤثرات المنطقية .

```
(count <= 100) AND (ch1 <> 'A')
(balance < 1000.0) OR (status = 'R')
(answer < 0) OR ((answer > 5.0) AND (answer < 10.0))
(pay >= 1000.0) AND (NOT single)
```

لاحظ أن ch1 ، status متغيران حرفيان فى هذه الأمثلة ، كما أن single هو متغير بوليان . أما بقية المتغيرات ، فهى عددية ( إما حقيقية أو صحيحة ) .

لاحظ أيضا أن عناصر بوليان موضوعة بين قوسين ، لتجنب أى غموض فى ترتيب التنفيذ .

تذكر أيضا أنه هناك نوعين أساسيين من العبارات فى البسكال ، وهما : النوع البسيط simple والنوع المرتب structured ( انظر قسم ٢ من الفصل الحادى عشر ) تشير العبارات البسيطة إلى عبارات تحديد وأدلة إجراءات أو عبارات GOTO . وسبق أن ناقشنا بالفعل عبارات التحديد ودلائل الإجراءات فى قسم ٢ من الفصل الثالث ، وقسم ٢ من الفصل الثانى عشر . وسوف تناقش عبارات GOTO فى نهاية هذا الفصل .

والعبارات المرتبة مهمة جدا فى هذا الوقت ، وذلك بسبب وقوع مكونات التحكم فى البسكال فى هذه الفئة . وسوف تقدم مناقشة تفصيلية لكل من هذه المكونات فيما بعد فى هذا الفصل . دعنا على أية حال نتذكر الآن المناقشة السابقة لأحد أنواع العبارات المرتبة ، وهى العبارة المركبة .

تحتوى العبارة المركبة compound statement على تتابع من اثنين أو أكثر من العبارات المتتالية والمحصورة بين الكلمتين الأساسيتين BEGIN , END ، مع فصلها عن بعضها بواسطة فواصل منقوطة . والعبارات الفردية التى تتكون منها العبارة المركبة يمكن أن تكون هى نفسها بسيطة أو مرتبة . وعلى هذا ... فإن العبارة المرتبة الواحدة ( وهى مكونات التحكم ) يمكن أن تتواجد داخل عبارة مركبة . والأكثر من ذلك ... يمكن أن ترحل العبارة المركبة داخل عبارة مركبة أخرى .

## مثال (٦-٣)

فيما يلي عبارة مركبة واحدة سبق لنا رؤيتها من قبل :

```
BEGIN
  read(radius);
  area := 3.14159*sqr(radius);
  write(radius,area)
END
```

وفيما يلي عبارة مركبة أكثر تعقيدا :

```
BEGIN
  sum := 0;
  FOR count := 1 TO n DO
    BEGIN
      read(x);
      sum := sum+x
    END;
  write(' sum = ',sum)
END
```

لاحظ أن العبارة المركبة الثانية تحتوي على مكون FOR للتحكم . ( سوف نناقش تفاصيل مكون التحكم هذا فيما بعد في هذا الفصل - انظر قسم ٤ من هذا الفصل ) . لاحظ أيضا أن العبارة المركبة الأصغر تم ترحيلها داخل مكون FOR للتحكم . وعلى هذا ... فإننا نرى مثالا لعبارة مركبة مرحلة داخل مكون تحكم موجود داخل عبارة مركبة أخرى .

مكونات التحكم الموجودة في هذا الفصل تستخدم تعبيرات بوليان ، والعبارات المركبة استخداما واسعا .

## ٢ - مكون « بينما ... أعمل » : 2. THE WHILE-DO STRUCTURE

مكون WHILE - DO هو مكون تحكم تكرارى يستخدم لإجراء نورات شرطية . والصيغة العامة لهذا المكون هي :

*WHILE boolean expression DO statement*

جزء العبارة من المكون يتكرر تنفيذه ، طالما أن قيمة تعبير بوليان تظل صحيحة . ويمكن أن تكون هذه العبارة بسيطة أو مرتبة ، بالرغم من أنها عادة ماتكون عبارة مركبة تحتوي على بعض المعالم التي يمكنها أن تبدل من قيمة تعبير بوليان .

افترض على سبيل المثال أننا نريد أن نكتب الأرقام من 1 إلى 20 على أن يكون هناك رقم واحد في كل سطر . يمكن تحقيق ذلك بمكون WHILE - DO التالى :

```
digit := 1;
WHILE digit <= 20 DO
  BEGIN
    writeln(digit);
    digit := digit + 1
  END;
```

حيث افترض أن digit متغير صحيح . وعلى هذا ... فإننا نبدأ بقيمة لهذا المتغير مساوية ١ . ونستمر في كتابة القيمة الحالية للمتغير digit بزيادة قيمته بمقدار ١ ، ثم تكرار نفس الدورة .

التأثير النهائي لمكون WHILE - DO هذا هو تكرار عملية الكتابة ، وزيادة القيمة في النهاية 20 مرة ، لينتج عن ذلك 20 سطرا متتاليا من المخرجات ، يحتوى كل سطر على قيمة صحيحة تبدأ بالرقم 1 في السطر الأول ، وتنتهى بالعدد 20 في السطر الأخير .

وبالمثل افترض أننا نريد أن نحدد مجموع أول عدد  $n$  من الأرقام ، حيث  $n$  هو متغير صحيح . يمكن أن يتحقق

ذلك بكتابة مايلي :

```
sum := 0;
digit := 1;
WHILE digit <= n DO
  BEGIN
    sum := sum + digit;
    digit := digit + 1
  END;
```

كما أن الكتابة التالية متكافئة مع الكتابة السابقة :

```
sum := 0;
digit := 1;
WHILE digit < n+1 DO
  BEGIN
    sum := sum + digit;
    digit := succ(digit)
  END;
```

حيث  $sum$  ,  $digit$  وكذلك  $n$  متغيرات صحيحة .

في أى حالة من الحالتين نبدأ بقيمة تساوى 0 للمتغير  $sum$  ، ثم نستمر في إضافة قيم متتالية لـ  $digit$  الى  $sum$  . وتستمر عملية إضافة القيمة الحالية للمتغير  $digit$  الى  $sum$  . وبعد ذلك زيادة قيمة  $digit$  بمقدار 1 ، طالما أن القيمة الحالية للمتغير  $digit$  أقل من  $n+1$  . وفي نهاية هذه العملية يقدم  $sum$  مجموع أول  $n$  رقما أى :

$$1 + 2 + 3 + \dots + n.$$

ويستخدم مكون WHILE-DO بكثرة في البسكال ، حيث يوجد هناك العديد من تطبيقات البرمجة التي تتطلب هذا النوع من إمكانية الدورات الشرطية . وسوف نرى العديد من عينات البرامج التي تستخدم مكون WHILE-DO خلال هذا الكتاب .

### مثال (٦-٤)

حساب متوسط قائمة من الأعداد . دعنا نستخدم مكون WHILE-DO في الحصول على متوسط قائمة من الأعداد . وتعتمد هذه الطريقة على استخدام المجموع الجزئي الذي توضع له قيمة ابتدائية مساوية للصفر ، وبعد ذلك يتم تجديده كلما حدثت قراءة لعدد جديد داخل الكمبيوتر . وعلى هذا ... فإن المشكلة تحتاج إلى تكرار .

يمكن أداء الحسابات الفعلية كما يلي :

(١) تحديد قيمة تساوى 1 للمتغير الصحيح Count (يستخدم هذا المتغير كعداد للدورة)

(٢) تحديد قيمة تساوى 0 للمتغير الحقيقي Sum .

(٣) قراءة قيمة  $n$  .

(٤) تكرار أداء الخطوات التالية ، طالما أن قيمة العداد لا تتعدى  $n$  ( أى أثناء احتواء المتغير Count على قيمة أقل من  $n+1$  ) .

(١) قراءة أحد الأعداد الموجودة في القائمة ( يمثل كل عدد بمقياس حقيقي x ) .

(ب) إضافة العدد إلى قيمة Sum .

(ج) زيادة قيمة Count بمقدار 1 .

(هـ) قسمة قيمة Sum على n للحصول على المتوسط المطلوب .

(٦) كتابة القيمة المحسوبة للمتوسط .

وفيما يلي برنامج البسكال الذي يؤدي ذلك .

```
PROGRAM averagel(input,output);

(* This program calculates the average of n numbers
   using a WHILE - DO structure. *)

VAR n,count : integer;
    x,sum,average : real;

BEGIN (* action statements *)
  count := 1;
  sum := 0;
  readln(n);
  WHILE count < n+1 DO
    BEGIN
      readln(x);
      sum := sum+x;
      count := count+1
    END; (* count < n+1 *)
  average := sum/n;
  writeln(' The average is ',average)
END.
```

لاحظ أن مكون WHILE-DO يحتوي على عبارة مركبة تتسبب في ازدياد قيمة Count ، وذلك بالإضافة إلى عبارات أخرى . وفي واقع الأمر ... فإن هذا يتسبب في جعل تعبير بولياني :

count < n+1

خاطئا في إحدى المرات ، وبالتالي يفصل البرنامج عن الكمبيوتر .

لاحظ أيضا أن الدورة لن تنفذ على الإطلاق ، إذا ما حددت قيمة أقل من واحد للمتغير n . وبالطبع هذا شيء واقعي .

لاحظ أخيرا أن العبارة المركبة الداخلية جدا تم ترحيلها داخل مكون WHILE-DO . ويتسبب ذلك في جعل قراءة مكون WHILE-DO سهلة .

### ٣ - مكون « كرر .. حتى » : 3. THE REPEAT-UNTIL STRUCTURE

مكون REPEAT - UNTIL هو مكون تحكم للتكرار ، يستخدم لإجراء دورة شرطية . وهو شبيه بمكون WHILE-DO . ويكمل هذان المكونان بعضهما البعض في بعض المواقف . والصيغة العامة لمكون REPEAT - UNTIL هي :

REPEAT sequence of statements UNTIL boolean expression

يتكرر تنفيذ مجموعة العبارات حتى يتحقق شرط بوليان . لاحظ أن مجموعة العبارات تنفذ مرة واحدة على الأقل دائما ، حيث إن تعبير بوليان لا يتم اختباره إلا عند الانتهاء من مكون التحكم . ( وهذا على عكس مكون WHILE-DO الذى لا ينفذ على الإطلاق إذا كانت القيمة الابتدائية لتعبير بوليان خاطئة ) .

لاحظ أن هذا المكون يسمح بوجود مجموعة عبارات ، بينما مكون WHILE-DO لا يسمح إلا بعباراة واحدة فقط ( ومن الممكن أن تكون عبارة مركبة ) . ومجموعة العبارات الموجودة داخل مكون التحكم REPEAT - UNTIL ليست فى حاجة لأن توضع بين BEGIN , END . وعلى هذا ... فإن الكلمتين الرئيسيتين REPEAT - UNTIL تعملان عمل الأقواس التى تحدد بداية ونهاية تسلسل العبارة . ومن الممكن أن توجد جملة مركبة أو مكون تحكم آخر بالطبع داخل مجموعة الجمل . بالإضافة إلى ذلك ... عادة ماتحتوى مجموعة العبارات بعض معالم يمكنها أن تغير من قيمة تعبير بوليان .

ولتوضيح استخدام مكون Repeat - until اعتبر مرة أخرى مشكلة كتابة الأرقام من 1 إلى 20 ، على أن يكتب كل رقم على سطر منفصل . سبق أن رأينا بالفعل كيف يمكن تحقيق ذلك باستخدام مكون WHILE - DO ( انظر قسم ٢ من هذا الفصل ) . والآن دعنا نستخدم مكون REPEAT - UNTIL لنفس الغرض .

```
digit := 1;
REPEAT
    writeln(digit);
    digit := digit + 1
UNTIL digit > 20;
```

مرة أخرى فإن digit يكون متغيرا صحيحا

نبدأ مرة أخرى بقيمة تساوى 1 للمتغير digit ، ثم نستمر فى كتابة القيمة الحالية للمتغير digit ، ونزيد قيمته بمقدار 1 ، ثم تعاد نفس الدورة مرة أخرى . وتستمر العملية حتى UNTIL تتعدى القيمة الحالية للمتغير digit ( والذى يكون قد ازدادت قيمته توا ) القيمة 20 .

والتأثير النهائى لمكون REPEAT - UNTIL هو نفس التأثير المناظر لمكون WHILE - DO سالف الذكر . وعلى هذا ... ينتج 20 سطرا من أسطر المخرجات المتتالية ، وفى كل سطر تظهر قيمة أحد الأرقام المتتالية .

دعنا نعتبر الآن استخدام مكون REPEAT - UNTIL فى تحديد مجموع أول عدد n من الأرقام ، حيث n هو متغير صحيح معروف ( انظر قسم ٢ من هذا الفصل ) . المكون المطلوب هو :

```
sum := 0;
digit := 1;
REPEAT
    sum := sum + digit;
    digit := succ(digit)
UNTIL digit > n;
```

مرة أخرى يكون sum , digit متغيرين صحيحين

وفى هذه الحالة نبدأ بقيمة تساوى صفر للمتغير sum ، ثم نستمر فى إضافة قيم متتالية للمتغير digit الى المتغير sum . وتستمر عملية إضافة القيمة الحالية للمتغير digit إلى sum ، ثم زيادة قيمة المتغير digit بمقدار 1 حتى UNTIL تتعدى القيمة الحالية للمتغير digit مقدار n . عند ذلك يمثل sum مجموع أول n من الأرقام ، أى يمثل  $1 + 2 + 3 + \dots + n$  .

ويستخدم مكون REPEAT - UNTIL كما يستخدم مكون WHILE - DO بكثرة في البسكال . وأحيانا يكون اختيار أحد هذين المكونين بسبب الأفضلية الشخصية فقط . وفي تطبيقات أخرى يتأثر الاختيار بالرغبة في اختبار تعبير بولياني ، إما في بداية مكون اختيار التحكم أو في نهايته .

### مثال (٦-٥)

حساب متوسط قائمة من الأعداد . دعنا نستخدم الآن مكون REPEAT - UNTIL للحصول على متوسط قائمة بالأعداد . وهذه هي نفس المشكلة المذكورة في مثال ٦ - ٤ مع استخدام WHILE - DO . وعلى هذا ... فإن الطريقة تشبه طريقة مثال ٦ - ٤ ، مع السماح لمكونات تحكم مختلفة .

يمكن إجراء الحسابات الفعلية كمايلي :

(١) تحديد قيمة تساوي 1 للمتغير الصحيح count ( يستخدم هذا المتغير كعداد للدورة ) .

(٢) تحديد قيمة تساوي 0 للمتغير الحقيقي sum .

(٣) قراءة قيمة n .

(٤) تكرار أداء الخطوات التالية حتى تتعدى قيمة count القيمة n

(أ) قراءة قيمة أحد الأعداد الموجودة في القائمة ( يمثل كل عدد بمتغير حقيقي x ) .

(ب) إضافة العدد إلى قيمة sum .

(ج) زيادة قيمة count بمقدار 1 .

(هـ) قسمة قيمة sum على n للحصول على المتوسط المطلوب .

(٦) كتابة القيمة المحسوبة للمتوسط .

لاحظ التشابه بين هذا التخطيط والتخطيط المقدم في مثال ٦ - ٤

وبرنامج البسكال المناظر لذلك هو مايلي :

```
PROGRAM average2(input,output);

(* This program calculates the average of n numbers
   using a REPEAT - UNTIL structure. *)

VAR n,count : integer;
    x,sum,average : real;

BEGIN  (* action statements *)
    count := 1;
    sum := 0;
    readln(n);
    REPEAT
        readln(x);
        sum := sum+x;
        count := count+1
    UNTIL count > n;
    average := sum/n;
    writeln(' The average is ',average)
END.
```

لاحظ أن مجموعة العبارات المحصورة في مكون REPEAT , UNTIL تحتوي على عبارة تتسبب في زيادة قيمة count . وهذا يتسبب - في واقع الأمر - في أن يصبح تعبير بوليان  $count > n$  صحيحا ليفصل البرنامج عن الكمبيوتر .

ويجب أن يكون مفهوما أن هذه الدورة تنفذ دائما مرة واحدة على الأقل ، حيث إن قيمة تعبير بوليان لاختبر إلا في نهاية الدورة .

لاحظ أخيرا أن مجموعة العبارات الموجودة داخل مكون REPEAT - UNTIL مرحلة للداخل ، وذلك لتعريف مدى المكون بسهولة .

ويجب أن يقارن هذا المثال بدقة مع مثال ٦ - ٤ . وكل من المكونين ، سواء أكان مكون WHILE - DO أم مكون REPEAT - UNTIL صحيحان بالنسبة لهذه المشكلة . وفي العديد من الحالات - على أية حال - فإن طبيعة المشكلة تقترح أولوية لأحد هذين المكونين على الآخر .

#### ٤ - مكون FOR : 4. THE FOR STRUCTURE

يستخدم مكون FOR لأداء دورات غير شرطية في البسكال ، أي أن هذا المكون يسمح بتكرار بعض الإجراءات عددا محددا من المرات .

ومكون FOR له صيغتان ، الصيغة الأكثر شيوعا ، هي :

`FOR control variable := value 1 TO value 2 DO statement`

يمكن أن يكون جزء العبارة من المكون بسيطا أو مركبا ، وذلك بالرغم من أن العبارة المركبة يمكنها أن تحتوي مكونات تحكم أخرى . وتنفذ هذه العبارة لكل قيمة من القيم المتتالية المحددة لتغيير التحكم . وعدد القيم المحددة لتغيير التحكم يحدد على ذلك عدد مرات تنفيذ العبارة .

ويجب أن يكون متغير التحكم من النوع البسيط من أي نوع من أنواع المتغيرات ، باستثناء النوع الحقيقي . وعادة مايكون متغيرا صحيحا ، أو متغيرا يقوم بتعريفه المستفيد . ( وسوف نناقش المتغيرات التي يعرفها المستفيد في الفصل الثامن من الكتاب ) . وتحدد قيمة ابتدائية 1 value في البداية لتغيير التحكم . ويأخذ متغير التحكم القيمة المحددة بواسطة 1 value في البداية . ويأخذ متغير التحكم القيمة التالية تلقائيا في كل مرة تتكرر العبارة ، وحتى يأخذ القيمة المحددة بانها 2 value . وإذا كان متغير التحكم من النوع الصحيح ، فسوف تزداد قيمته تلقائيا بمقدار 1 في كل مرة يتم فيها تنفيذ العبارة عدد مرات يساوي الرقم الناتج من باقى طرح  $value 1 + 1$  من  $value 2$  .

واتوضيح مكون FOR ، دعنا نعتبر مشكلة كتابة أول 20 رقم صحيح موجبة مرة أخرى ، مع كتابة كل رقم على سطر منفصل . ( سبق أن رأينا كيف يمكن أداء ذلك باستخدام مكون WHILE - DO في قسم ٢ - ، وباستخدام مكون REPEAT - UNTIL في قسم ٢ - ) . ونحتاج إلى عبارة واحدة فقط FOR - TO لأداء هذا النشاط . وبصفة خاصة يمكننا كتابة مايلي :

`FOR digit := 1 TO 20 DO writeln(digit);`

حيث يكون digit متغيرا صحيحا .

يأخذ المتغير digit في هذا المثال القيم المتتالية 1 , 2 , 3 , ... , 20 ، متسببا في تنفيذ الدورة 20 مرة . وأثناء كل مرة من هذه المرات تكتب القيمة الحالية للمتغير digit على سطر منفصل ، كما هو مطلوب تماما .

وبالمثل فإن مشكلة تحديد مجموع أول n من الأرقام ، يمكن أن يعبر عنها كما يلي :

`sum := 0;      sum := sum + digit;`



وهنا نبدأ بتحديد قيمة 0 للمتغير sum . وفى كل مرة تنفذ فيها الدورة تضاف القيمة الحالية الموجودة فى digit إلى sum . وعلى هذا ... فإن sum يمثل القيمة المطلوبة ، وهى :

$$1 + 2 + 3 + \dots + n$$

بعد تكرار تنفيذ الدورة عدد n من المرات .

لاحظ السهولة الكبيرة الناتجة من استخدام مكون FOR ، وذلك بالمقارنة بكل من مكونى DO - WHILE و REPEAT - UNTIL بالنسبة لهذه المشاكل ( انظر قسم ٢ - ، وقسم ٣ - ) . وعادة ماتكون هذه هى الحالة عندما يكون عدد مرات تكرار الدورة معروفا مسبقا .

وهناك قواعد محددة يجب اتباعها فى كتابة عبارة FOR . يمكن أن يعبر عن القيم Value 1 , Value 2 بصفة خاصة بثوابت أو متغيرات وتعبيرات . وعلى أية حال ... يجب أن تكون هاتان القيمتان فى نفس النوع مثل نوع متغير التحكم . كما يجب أن تكون قيمة Value 1 أقل من قيمة Value 2 إذا ماكان مطلوباً تكرار العبارة أكثر من مرة واحدة . ( إذا كانت قيمة Value 1 مساوية لقيمة Value 2 ، فتتفاد العبارة مرة واحدة فقط ، أما إذا كانت قيمة Value 1 أكبر من قيمة Value 2 ، فلن تنفذ العبارة على الإطلاق ) .

مثال (٦ - ٦)

إن استخدام مكون FOR أقل تعقيدا فى واقع الأمر عما يبدو من أول مرة . ويتضح ذلك من المثالين التاليين :

```
(a)      sum := 0;
          FOR count := 1 TO n DO BEGIN readln(x); sum := sum+x END;
          writeln(' sum=',sum);

(b)      sum := 0;
          FOR count := n TO (3*n+1) DO
            BEGIN
              readln(x);
              sum := sum+x
            END;
          writeln(' sum=',sum);
```

يحتوى المثال الأول على عبارة مركبة تنفذ عدد n من المرات . ( لاحظ أن n متغير صحيح ، يفترض أن قيمته معروفة ، وأن x متغير حقيقى ) . وخلال كل مرة من مرات تنفيذ الدورة ، يقرأ عدد جديد ( قيمة جديدة للمتغير x ) داخل الكمبيوتر ، ويضاف إلى sum . وبعد ذلك يكتب مجموع كل الأعداد بعد الانتهاء التام من تكرار الدورة .

فى المثال الثانى تعطى القيمة الابتدائية والقيمة النهائية لمتغير التحكم عن طريق متغير صحيح وتعبير صحيح على التوالى . لاحظ أن العبارة المركبة انتشرت الآن لتشغل عدة أسطر ، مع عمل الترحيل المناسب . ( هذه هى الصيغة المفضلة ) .

أما الصيغة الثانية لمكون FOR ، فتشبه الصيغة الأولى ، وذلك باستثناء أن الكلمة الأساسية DOWNTO محل الكلمة الأساسية TO . وعلى هذا ... يمكن كتابة مكون على النحو التالى :

FOR control variable := value 1 DOWNTO value 2 DO statement

والإجراء المتخذ بهذه الصيغة لمكون FOR يشبه الإجراء المتخذ بواسطة الصيغة الأولى ، فيما عدا أن متغير التحكم يتم تقويمه للخلف ، بدلا من تقويمه للأمام . وعلى هذا ... فإذا ماكان متغير التحكم من النوع الصحيح ، فسوف يقل تلقائيا بمقدار 1 ، وذلك من القيمة Value 1 إلى القيمة Value 2 أثناء التكرار المتتالى للدورة . وعلى هذا ... يجب أن تكون قيمة Value 1 أكبر من قيمة Value 2 . ( فإذا ماتساوت القيمتان ، تنفذ العبارة مرة واحدة فقط . أما إذا كانت قيمة Value 1 أقل من قيمة Value 2 ، فلن تنفذ العبارة على الإطلاق ) .

### مثال (٦-٧)

فيما يلي توضيحاً للصيغة الثانية لمكون FOR :

```
FOR i := 0 DOWNT0 -12 DO
  BEGIN
    z := 2*i+5;
    writeln(' i=',i, ' z=',z)
  END;
```

سوف يتسبب هذا المثال في طباعة 13 سطرا . يحتوى كل سطر على القيمة الحالية للمتغير الصحيح i ، تتبعها القيمة المناظرة من العلاقة  $z = 2i + 5$  . لاحظ أن القيم المتتالية للمتغير i تتناقص من  $i = 0$  في أول سطر إلى  $i = -12$  في آخر سطر .

ويجب أن يكون مفهوماً أن تقويم Value 1 , Value 2 يحدث مرة واحدة فقط قبل تنفيذ الدورة لأول مرة . وعلى هذا ... فيجب ألا يحاول القارئ أن يغير أى من هاتين القيمتين داخل الدورة . كما يجب أن يحتاط القارئ أيضا بأن لا يستخدم متغير الدورة بعد فصل مكون FOR ، أى بعد الانتهاء من تنفيذ مكون FOR ، حيث إنه لا يكون معرفاً بطريقة طبيعية .

### مثال (٦-٨)

حساب متوسط قائمة بالأعداد . دعنا نعود الى مشكلة حساب متوسط قائمة الأعداد التى سبق أن تعرضنا لها فى مثال ٦ - ٤ ، ومثال ٦ - ٥ . وسوف نستخدم الآن - على أية حال - مكون FOR لأداء أنشطة الدورة .

(١) تحديد قيمة تساوى 0 للمتغير الحقيقى sum .

(٢) قراءة قيمة للمتغير n .

(٣) أداء الخطوات التالية عدد n من المرات ( أى أن متغير التحكم تتراوح قيمته من 1 إلى n ) .

(أ) قراءة العدد التالى فى القائمة . ( يمثل كل عدد بمتغير حقيقى x ) .

(ب) يضاف هذا العدد إلى قيمة sum .

(٤) قسمة قيمة sum على n للحصول على المتوسط المطلوب .

(٥) كتابة القيمة المحسوبة للمتوسط .

لاحظ التشابه الموجود بين هذا التخطيط والتخطيطين السابقين الموجودين فى مثال ٦ - ٤ ، ومثال ٦ - ٥ . وفيما يلي برنامج البسكال كاملاً .

```
PROGRAM average3(input,output);

(* This program calculates the average of n numbers
   using a FOR structure. *)

VAR n,count : integer;
    x,sum,average : real;

BEGIN (* action statements *)
  sum := 0;
  readln(n);
  FOR count := 1 TO n DO
    BEGIN
      readln(x);
      sum := sum+x
    END;
  average := sum/n;
  writeln(' The average is ',average)
END.
```

لاحظ أن متغير التحكم count حددت له قيمة ابتدائية تساوى 1 . وتزداد هذه القيمة تلقائيا بمقدار 1 فى كل مرة يتم فيها تنفيذ الدورة ، وذلك حتى تصل قيمة count فى النهاية إلى n . وعلى هذا ... فإن الدورة تنفذ عدد n من المرات بالضبط . لاحظ أيضا أن مجموعة العبارات الموجودة داخل مكون FOR مرحلة للداخل . وهذا يسمح بتعريف المكون ومحتوياته بسهولة .

ويجب أن يقارن القارئ هذه الطريقة لحل المشكلة ، مع كيفية حل نفس المشكلة فى كل من مثالى ٦ - ٤ ، و ٦ - ٥ . ويناسب استخدام مكون FOR هذه المشكلة الخاصة عن استخدام أى من مكونى WHILE - DO أو REPEAT - UNTIL ، حيث إن عدد مرات تكرار الدورة معروف مسبقا . وهناك العديد من المشاكل على أية حال لا يكون هذا هو حالها ( انظر مثال ٦ - ٢٦ ، ومشكلة ٦ - ٤٥ فى نهاية هذا الفصل ) . فى مثل هذه الحالات قد يكون أى من مكونى WHILE - DO أو REPEAT - UNTIL مناسباً أكثر . وعلى هذا ... فلا يجب أن يستخلص القارئ تعليقاً عاماً عن المنفعة النسبية لمكونات التحكم هذه ، فيجب أن يتم تقويم كل تطبيق طبقاً لحالته الخاصة .

## ٥ - مكونات التحكم المتداخلة : 5. NESTED CONTROL STRUCTURES

يمكن أن تتداخل مكونات التحكم واحدا داخل الآخر . وليس هناك حاجة لأن يكون كل من مكون التحكم الداخلى والخارجى من نفس النوع ، إلا أنه من الضرورى أن يكون أحد المكونات موجودا بأكمله داخل المكون الآخر . ويكلمات أخرى ... يجب ألا يبدأ مكون تحكم داخل مكون آخر وينتهى بعد الانتهاء من المكون الثانى .

### مثال (٦-٩)

إعادة حساب متوسط قائمة الأعداد . افرض أننا نريد حساب المتوسط للعديد من قوائم الأعداد . إذا ما عرفنا مسبقا عدد القوائم التى سوف يجرى لها حساب المتوسط ، فيمكننا عند ذلك استخدام مكون FOR للتحكم فى عدد مرات تنفيذ دورة حساب المتوسط . ويمكن تحقيق حساب المتوسط الفعلى باستخدام أى مكون من مكونات التكرار الثلاثة ، وهى : WHILE - DO أو REPEAT - UNTIL أو FOR كما سبق توضيحها فى مثال ٦ - ٤ ، ومثال ٦ - ٥ ومثال ٦ - ٨ .

دعنا نستخدم على سبيل المثال مكون REPEAT - UNTIL فى حساب المتوسط ، كما فى حالة مثال ٦ - ٥ وعلى هذا ... فإننا نستمر طبقاً لما يلى :

(١) قراءة قيمة للمتغير loopmax ، وهى قيمة صحيحة تحدد عدد القوائم التى سوف يحسب المتوسط لها .

(٢) قراءة متكررة لقائمة أعداد ، وتحديد متوسطها ( أى حساب متوسط للقيم المتتالية لمتغير الدورة loop التى تتراوح من 1 إلى loopmax ) . وتتبع نفس الخطوات المعطاه فى مثال ٦ - ٥ لحساب المتوسط .

وفيما يلي برنامج البسكال الكامل لأداء ذلك .

```
PROGRAM average4(input,output);

(* This program uses nested control structures
   to repeatedly calculate the average of n numbers *)

VAR loop,loopmax,n,count : integer;
    x,sum,average : real;

BEGIN (* action statements *)
  readln(loopmax);
  FOR loop := 1 TO loopmax DO
    BEGIN
      count := 1;
      sum := 0;
      readln(n);
      REPEAT
        read(x);
        sum := sum+x;
        count := count+1
      UNTIL count > n;
      average := sum/n;
      writeln(' list number ',loop,' The average is ',average)
    END (* loop *)
  END.
```

لاحظ طريقة تداخل مكون REPEAT - UNTIL داخل مكون FOR ( وهذا سهل رؤيته بسبب الترحيل للداخل ) وعلى هذا ... فإن عملية حساب المتوسط تنفذ بالكامل أثناء المرور خلال الدورة الخارجية .

ويتطلب الكثير من التطبيقات استخدام مكون FOR . في مثل هذه الحالات يجب أن يستخدم كل مكون OR متغير تحكما مختلفا . ( تذكر أيضا أن كل مكون FOR داخلي يجب أن يكون كله محصورا داخل مكون OR الخارجى ) . والمثال التالى يوضح ذلك .

مثال (٦-١٠)

عوامل الفائدة المركبة : نفرض أن  $P$  هي مجموع رأس المال المستثمر فى  $n$  من السنوات ، بمعدل فائدة  $i$  فى المئة فائدة مركبة سنوياً .  $F$  مقدار رأس المال المتجمع بعد  $n$  من السنوات ، ويمكن حسابها بواسطة المعادلة المعروفة التالية :

$$F = P(1 + i/100)^n$$

( وتعرف أيضاً بقانون الفائدة المركبة )

– وسيتم تنفيذ الحسابات بالطريقة التالية : –

- ١- تخصيص القيمة "صفر" للمتغير الحقيقي sum .
- ٢- اقرأ قيمة n .
- ٣- أداء الخطوات التالية n من المرات (بمعنى القيم المتتالية لعداد متغير التحكم فى المدى من ١ ← n .
- ١ - اقرأ العدد التالى فى القائمة ( كل عدد يمثل بالمتغير الحقيقي x ) .
- ب - اجمع هذا العدد على قيمة المتغير sum .
- ٤- إقسم قيمة sum على n للحصول على المتوسط المطلوب.
- ٥- أكتب القيمة المحسوبة للمتوسط.

وفيما يلى برنامج البسكال كاملا :

```
PROGRAM compoundinterest(output);
(* This program generates a table of compound interest factors
to calculate a future value, given a present value or vice versa *)
VAR i,n: integer;
    factor: real;
BEGIN (* action statements *)
    write(' n      5%      6%      7%      8%');
    writeln('      9%      10%     11%     12%');
    writeln;
    FOR n:=1 TO 20 DO
        BEGIN (* generate successive rows *)
            write(n:3);
            FOR i:= 5 TO 12 DO
                BEGIN (* generate entries within each row *)
                    factor:= exp(n*ln(1+0.01*i));
                    write(factor:9:5);
                END;
            writeln;
        END
    END.
```

لاحظ الطريقة التى استخدم بها البرنامج مكون FOR . يتحكم مكون FOR الخارجى فى عدد المضروبوات التى تحسب ، بينما يستخدم مكون FOR الداخلى فى حساب كل مضروب من هذه المضروبوات . لاحظ أن المكون الداخلى موجود كله داخل المكون الخارجى ، وأن كل من المكونين له متغير التحكم الخاص به . وعلى أية حال ... تستخدم القيمة الحالية لمتغير التحكم الخارجى (n) كقيمة نهائية لمتغير التحكم الداخلى . وتتطلب طبيعة المشكلة أن يستخدم n بهذه الطريقة .

عند تشغيل البرنامج يجب أن يكون هناك حرص بالنسبة لأقصى قيمة تحدد للمتغير n ، حيث إن المضروبوات يمكن أن تكون كبيرة جدا ، حتى بالنسبة لقيم متواضعة من قيم n . فإذا ما حددت قيمة كبيرة للمتغير n ، فقد يحدث سريان زائد overflow عند حساب مضروب n . لاحظ بصفة خاصة أن مضروب 8 هو 40.320 . وقد تكون هذه الكمية كبيرة جدا بالنسبة لبعض أجهزة الكمبيوتر الصغيرة .

## 6. THE IF STRUCTURE

٦ - مكون إذا :

مكون IF هو مكون تحكم شرطى ، يسمح باتخاذ بعض الإجراءات إذا أخذ أحد الشروط المنطقية قيمة معينة ( صحيح أو خاطئ ) .

وهذا المكون له صيغتان مختلفتان . وأبسط صيغة هي :

IF boolean expression THEN statement

وعادة ما يشار إلى هذا المكون بأنه مكون IF - THEN . وينفذ جزء العبارة من المكون إذا ما كان تعبير بوليان صحيحا فقط . أما إذا كان تعبير بوليان خاطئا ، فإن جزء العبارة من المكون لا ينفذ . ويمكن للعبارة نفسها أن تكون بسيطة أو مرتبة ، بالرغم من أنها عادة ما تكون مركبة .

مثال (٦-١١)

فيما يلي بعض أمثلة لمكون IF - THEN

```
IF count <= 100 THEN count := count+1;
IF tag = '+' THEN
  BEGIN writeln(accountno); credit := 0 END;
IF test THEN BEGIN x := 100; test := false END;
IF (balance < 1000.0) OR (status = 'R') THEN writeln(balance);
```

المتغير الصحيح count فى المكون الأول يزداد بمقدار 1 إذا لم تتعد القيمة الحالية (100) . ويتسبب المثال الثانى فى طباعة قيمة المتغير accountno ، وتحديد القيمة 0 للمتغير credit إذا ما حددت نجمة "\*" للمتغير الحرفى tag . لاحظ أن هذا المثال يحتوى على عبارة مركبة .

ويحتوى المثال الثالث على متغير بوليان ( test ) وعبارة مركبة . فإذا ما كانت القيمة الابتدائية للمتغير test صحيحة ، فتحدد القيمة 100 للمتغير x ، ويصبح test خاطئا .

وفى آخر مثال تطبع قيمة balance إذا ما كانت أقل من 1000.0 ، أو إذا ما كان المتغير الحرفى status يمثل الحرف R ( أو إذا ما تحقق الشرطان ) .

وأخيرا ... لاحظ أنه لا توجد فواصل منقوطة داخل مكون IF - THEN ، فيما عدا الفواصل الموجودة داخل العبارة المركبة .

والصيغة الثانية لمكون IF هي :

IF boolean expression THEN statement 1 ELSE statement 2

وعادة ما يشار إليها بأنها مكون IF - THEN - ELSE . وفى هذه الحالة تنفذ statement 1 إذا ما كان تعبير بوليان صحيحا ، وإلا فتنفذ statement 2 . لاحظ تنفيذ إحدى العبارتين دائما ( ومن المستحيل تنفيذهما معا ) . مرة أخرى ... العبارة الفردية يمكن أن تكون بسيطة أو مركبة . وعادة ما تكون العبارتان مركبتين .

يجب ألا تظهر الفواصل المنقوطة داخل مكون IF - THEN - ELSE ، إلا إذا استخدمت كفواصل داخل عبارة مركبة . وأحيانا يخطئ المبتدئون بوضع فواصل منقوطة قبل الكلمة الأساسية ELSE . ويجب تجنب ذلك ، حيث إن هذا يتسبب في خطأ في الترجمة .

### مثال (٦-١٢)

فيما يلي عدة أمثلة توضح استخدام مكون IF - THEN - ELSE .

```
IF status='S' THEN tax := 0.20*pay ELSE tax := 0.14*pay;

IF tag = '*' THEN BEGIN writeln(accountno); credit := 0 END
ELSE credit := 1000;

IF circle THEN BEGIN
    readln(radius);
    area := 3.141593*sqr(radius);
    writeln(' Area of circle=',area)
END
ELSE BEGIN
    readln(length, width);
    area := length*width;
    writeln(' Area of rectangle=',area)
END;
```

تحدد قيمة المتغير tax في المثال الأول بإحدى طريقتين ، طبقا للقيمة المحددة للمتغير الحرفي status .

ينظر المثال الثاني إلى الحسابات التي لها اشارات tagged . فإذا كان الحساب له اشارة ( أى إذا ماحدثت نجمة "\*" للمتغير الحرفي tag ) ، فيطبع رقم الحساب ويوضع حد المديونية مساويا للصفر ، وإلا فيحدد حد المديونية بأنه 1000 .

ويبين المثال الثالث كيفية حساب مساحة لأي من شكلين هندسيين ، فإذا كان متغير بولييان circle صحيحا ، تقرأ قيمة نصف القطر داخل الكمبيوتر ، ثم تحسب المساحة وتطبع بعد ذلك . أما إذا كان متغير بولييان circle خاطئا ، يقرأ طول وعرض المستطيل داخل الكمبيوتر ، وتحسب المساحة ، ثم تطبع قيمتها .

لاحظ مرة أخرى أنه لا توجد فواصل منقوطة في مكون IF - THEN - ELSE ، فيما عدا الفواصل الموجودة داخل العبارات المركبة .

مكونات IF يمكنها أن تتداخل مع بعضها ، وذلك مثل أى مكونات تحكم أخرى . وفيما يلي بعض الصيغ التي يمكن أن تأخذها مكونات IF في تداخلها مع بعضها .

الصيغة الأكثر شيوعا هي تداخل الطبقتين ، والتي تأخذ الشكل التالي :

```
IF be1 THEN IF be2 THEN s1 ELSE s2
ELSE IF be3 THEN s3 ELSE s4
```

حيث be 1 , be 2 , be 3 تمثل تعبيرات بولييان s 1 , s 2 , s 3 , s 4 تمثل عبارات . في هذه الحالة ينفذ مكون IF - THEN - ELSE كاملا إذا ماكانت be1 صحيحة . وينفذ مكون آخر إذا ماكانت be1 خاطئة . ومن الممكن بالطبع أن تحتوى s 1 , s 2 , s 3 , s 4 على مكونات IF - THEN - ELSE أيضا . وعلى هذا ... فيمكن أن يكون لدينا تداخل متعدد .

وفيما يلي صيغة أخرى لتداخل الطبقتين :

```
IF be1 THEN s1
    ELSE IF be2 THEN s2 ELSE s3

IF be1 THEN IF be2 THEN s1 ELSE s2
    ELSE s3

IF be1 THEN IF be2 THEN s1 ELSE s2
```

في كل من الحالة الأولى والثانية تتحدد طبيعة التبعية لمكون IF - THEN - ELSE بواسطة السطر المكتوبة عليه . وفي الحالة الأخيرة ليس من الواضح أى تعبير بوليان هو المصاحب لجزء ELSE . النتيجة هي bc 2 . وعلى هذا ... فإن المثال الأخير يكافئ مايلي :

```
IF be1 THEN
    BEGIN IF be2 THEN s1 ELSE s2 END
```

إذا ما أردنا أن يصاحب جزء ELSE تعبير بوليان bc1 ، بدلا من تعبير بوليان bc 2 ، فيجب أن نكتب مايلي :

```
IF be1 THEN BEGIN IF be2 THEN s1 END
    ELSE s2
```

وعلى هذا ... فيجب إجراء هذا النوع من التداخل بدقة ، وذلك لتجنب أى إبهام ممكن

مثال (٦-١٣)

إعداد شفرة للرموز . دعنا نكتب برنامج بسكال بسيط يقرأ رموز بشفرة ASCII ، ويخرج شفرة الرموز في مواقعها . فإذا كان الرمز حرفا أو رقما ، فإنه يحل محله بالرمز التالي له ، فيما عدا الحرف Z الذي يجب أن يحل محله A ، والحرف z الذى يحل محله a ، والرقم 9 الذى يحل محله 0 . وعلى هذا ... فإن 1 تصبح 2 ، C تصبح D ، p تصبح q وهكذا . وأى رمز غير الحروف والأرقام تحل محله نجمة (\*). واستمر في الحسابات حتى يتم إدخال النجمة كرمز مدخلات ، ( يجب أن تفسر النجمة على أنها تحقق لشرط التوقف ) .

وحتى يعد تخطيطا للبرنامج ، دعنا نعرف المتغيرات التالية :

charin = متغير حرفي يمثل رمز المدخلات .

charout = متغير حرفي يمثل رمز المخرجات .

flag = متغير بوليان يحدد ما إذا كانت إجراءات الدورة تستمر أم لا .

وتستمر الحسابات كمايلي :

(١) تحديد قيمة صحيح للمتغير flag .

(٢) تكرار الخطوات التالية ، حتى تصبح قيمة flag خاطئاً .

(أ) قراءة قيمة للمتغير charin .

(ب) إذا كانت قيمة charin هي النجمة (\*) ، توضع قيمة خطأ للمتغير flag ، وإلا فيتحدد إذا ما كان charin يمثل حرفا أم رقما .

(١) إذا كان charin يمثل حرفا أو رقما ؛ ضع القيمة المناسبة للمتغير charout .

(٢) إذا كان charin يمثل رمزا خاصا ، ولا يمثل حرفا أو رقما ؛ ضع نجمة (\*) للمتغير charout .



٣) أخرج القيمة الجديدة المحددة للمتغير charout .

وفيما يلي برنامج البسكال المناظر لذلك .

```
PROGRAM charactercode(input,output);

(* THIS PROGRAM REPLACES INPUT CHARACTERS
   WITH EQUIVALENT ENCODED CHARACTERS *)

VAR charin,charout : char;
    flag : boolean;
BEGIN
    flag := true;
    REPEAT
        write(' Enter character: ');
        readln(charin);
        IF charin = '*' (* stopping criterion *)
        THEN flag := false
        ELSE BEGIN (* character replacement *)
            IF ((charin >= '0') AND (charin < '9')) OR
                ((charin >= 'A') AND (charin < 'Z')) OR
                ((charin >= 'a') AND (charin < 'z'))
            THEN charout := succ(charin)
            ELSE IF charin = '9'
                THEN charout := '0'
                ELSE IF charin = 'Z'
                    THEN charout := 'A'
                    ELSE IF charin = 'z'
                        THEN charout := 'a'
                        ELSE charout := '*';
            writeln(' New character: ',charout);
            writeln
        END
    UNTIL flag = false
END.
```

لاحظ أن البرنامج يحتوى على العديد من عبارات IF - THEN - ELSE بمستويات تداخل متعددة . لاحظ أيضا أننا اخترنا استخدام مكون REPEAT - UNTIL لأداء عمليات التورية . وقد كان من الممكن اختيار مكون WHILE - DO ببساطة ، إلا أن مكون FOR ليس مناسباً في هذه الحالة .

وينتج عن تنفيذ هذا البرنامج المخرجات التالية . ( موضوع خط تحت استجابات المستفيد ) .

```
Enter character: f
New character: g

Enter character: z
New character: a

Enter character: P
New character: Q

Enter character: 5
New character: 6

Enter character: 9
New character: 0

Enter character: ?
New character: *

Enter character: *
```

مثال (٦-١٤)

حل معادلة جبرية . بالنسبة المحبين للرياضيات ، يوضح هذا المثال كيف يمكن استخدام الكمبيوتر فى حل معادلات جبرية لايمكن حلها بالطرق البدائية . أعتبر على سبيل المثال المعادلة التالية :

$$x^5 + 3x^2 - 10 = 0.$$

لايمكن إعادة ترتيب هذه المعادلة للحصول على حل دقيق للمتغير  $x$  . وعلى أية حال ... يمكننا أن نحدد الحل بتكرار عملية المحاولة والخطأ ( أى عملية تكرارية ) والتي تحسن بصفة دائمة من التقدير الأولي للحل .

ونبدأ بإعادة ترتيب المعادلة لتأخذ الشكل التالى :

$$x = (10 - 3x^2)^{1/5}$$

والعملية هى أننا نقدر قيمة المتغير  $x$  ، ونعوض بهذه القيمة فى الطرف الأيمن للمعادلة التى أعيد ترتيبها ، وذلك لحساب قيمة  $x$  الجديدة ، فإذا ماكانت قيمة  $x$  الجديدة مساوية لقيمة  $x$  القديمة (أو قريبة جدا منها ) ، نكون قد حصلنا على حل المعادلة ، وإلا فإننا نعوض بقيمة  $x$  الجديدة فى الطرف الأيمن للمعادلة ، لنحصل على قيمة أخرى للمتغير  $x$  وهكذا . وتستمر هذه العملية حتى يصبح هناك قيمتان متتاليتان للمتغير  $x$  قريبتان جدا من بعضهما ( أى حتى تبدأ الحسابات فى التقارب ) ، أوحتى يتم تنفيذ عدد محدد من التكرارات ( حتى لا يستمر تكرار الحسابات عددا طويلا من المرات إذا لم يحدث تقارب للتتائج المحسوبة ) .

ولرؤية كيفية عمل هذه الطريقة ، افرض أننا اخترنا قيمة ابتدائية 10 للمتغير  $x$  . بالتعويض بهذه القيمة فى الطرف الأيمن للمعادلة : نحصل على :

$$x = [10 - 3(10)^2]^{1/5} = 1.47577$$

ثم نعوض بعد ذلك بقيمة  $x$  الجديدة فى الطرف الأيمن للمعادلة لنحصل على :

$$x = [10 - 3(1.47577)^2]^{1/5} = 1.28225$$

وبالاستمرار فى هذه العملية ، فإننا نحصل على :

$$x = [10 - 3(1.28225)^2]^{1/5} = 1.38344$$

$$x = [10 - 3(1.38344)^2]^{1/5} = 1.33613$$

وهكذا . لاحظ أن القيم المتتالية للمتغير  $x$  بدأت تتقارب ناحية نتيجة نهائية .

ويعتمد نجاح الطريقة على قيمة التقدير الابتدائى . فإذا ما كانت هذه القيمة كبيرة جدا ، فإن الكمية الموجودة بين القوسين تكون سالبة ، ولايمكن رفع الرقم السالب إلى أس كسرى . وعلى هذا ... فيجب أن نختبر إذا ماكانت القيمة  $x^2 - 3 - 10$  سالبة أم لا عند التعويض بقيمة جديدة للمتغير  $x$  فى الطرف الأيمن للمعادلة .

ولكى نكتب تخطيطا للبرنامج ، دعنا نعرف الرموز التالية :

count = عدد مرات التكرار ( يتزايد count بمقدار وحدة واحدة فى كل مرة يحدث فيها التكرار ) .

guess = قيمة  $x$  التى يعوض بها فى الطرف الأيمن للمعادلة .

root = قيمة x المحسوبة الجديدة .

test = القيمة  $(10-3x^2)^{1/5}$  .

error = القيمة المطلقة للفرق بين root و guess .

flag = متغير بوليان يحدد ما إذا كان التكرار يستمر أم لا .

سوف نستمر في الحسابات ، حتى يتحقق أحد الشروط التالية :

(١) تصبح قيمة error أقل من 0.00001 ( وفي هذه الحالة فإننا نحصل على التقارب ) .

(٢) إتمام 50 تكرارا ( أى يصبح count = 50 ) .

(٣) تكون قيمة المتغير سالبة ( وفي هذه الحالة لا يمكن الحسابات أن تستمر ) .

دعنا نتقدم أكثر في الحسابات عن طريق كتابة كل قيمة للجذر كمخرجات .

والآن يمكننا أن نكتب التخطيط التالى للبرنامج :

(١) وضع قيمة ابتدائية للمتغيرين flag , count ( وضع قيمة صحيح للمتغير flag ، وقيمة 0 للمتغير count ) .

(٢) قراءة قيمة ابتدائية كتقدير للحل ( المتغير guess ) .

(٣) إجراء العمليات التكرارية التالية ، طالما أن قيمة flag هي صحيح :

(أ) زيادة خطوة العداد ( زيادته بمقدار 1 ) .

(ب) تحديد قيمة خطأ للمتغير flag إذا ما كانت القيمة الجديدة للمتغير count مساوية 50 . ( وهذا يحدد آخر مرة لتكرار الدورة ) .

(ج) تقويم test . فإذا ما كانت قيمته موجبة ، يستمر العمل على النحو التالى :

(١) حساب قيمة جديدة للمتغير root ، ثم كتابة قيمة count الحالية كمخرجات ، تليها قيمة root الحالية .

(٢) تقويم error ( القيمة المطلقة للفرق بين guess , root ) . فإذا ما كانت هذه القيمة أكبر من 0.00001 ، تحدد قيمة root للمتغير guess ، ويستمر العمل بتكرار الدورة ، وإلا فتكتب القيم الحالية لكل من count , root ، كمخرجات وتوضع قيمة flag خطأ ( وتعتبر القيمة الحالية للمتغير root أنها هي الحل المطلوب ) .

(د) أما إذا كانت قيمة test غير موجبة ، فلا يمكن عند ذلك استمرار الحسابات . وعند ذلك تكتب رسالة خطأ مناسبة ( "numbers out of range" ) ، وتوضع قيمة flag خاطئة .

(٤) عند إتمام عمليات الدورة ، تكتب رسالة خطأ مناسبة ( "Convergence not obtained" ) إذا ما كانت قيمة count 50 ، وكانت قيمة error أكبر من 0.00001 .

ودعنا الآن نغير عن تخطيط البرنامج في صورة شفرة شبيهة ، وذلك لتبسيط النقل من التخطيط العام إلى برنامج بسكال .

```

PROGRAM equation(input,output);

(* variable declarations *)

BEGIN

(* initialize flag and count *)

(* read guess *)

WHILE flag DO
  BEGIN
    (* increment count *)

    IF count = 50 THEN flag := false;

    (* evaluate test *)

    IF test > 0 THEN BEGIN (* another iteration *)

      (* evaluate root *)

      (* write count and root *)

      (* evaluate error *)

      IF error > 0.00001 THEN guess := root;
      ELSE BEGIN
        flag := false;

        (* write final answer -
        root and count *)

        END

      END

    ELSE BEGIN
      flag := false;
      (* write error message *)
    END

    END;
    IF (count = 50) AND (error > 0.00001) THEN
      (* write error message *)
    END.

```

وفيما يلي برنامج البسكال كاملا :

```

PROGRAM equation(input,output);

(* THIS PROGRAM DETERMINES THE ROOTS OF AN ALGEBRAIC EQUATION
  USING AN ITERATIVE PROCEDURE *)

```

(تكملة البرنامج في الصفحة التالية)

```

VAR count : integer;
    guess,root,test,error : real;
    flag : boolean;
BEGIN
    flag := true;  (* Initialize *)
    count := 0;
    write(' Initial guess= ');  (* Begin input routine *)
    readln(guess);
    writeln;
    WHILE flag DO  (* Begin main loop *)
        BEGIN
            count := count+1;
            IF count = 50 THEN flag := false;
            test := 10-3*sqr(guess);
            IF test > 0
                THEN BEGIN  (* Another iteration *)
                    root := exp(0.2*ln(test));
                    write(' Iteration number ',count:2);
                    writeln('  x= ',root:7:5);
                    error := abs(root-guess);
                    IF error > 0.00001
                        THEN guess := root
                        ELSE BEGIN  (* Write final answer *)
                            flag := false;
                            writeln;
                            write(' Root= ',root:7:5);
                            writeln('  No. of iterations= ',count:2)
                        END
                    END
                ELSE BEGIN  (* Error message *)
                    flag := false;
                    writeln;
                    write(' Numbers out of range - ');
                    writeln('try again using another initial guess')
                END
            END;
        IF (count = 50) AND (error > 0.00001) (* Error message *)
            THEN BEGIN
                writeln;
                writeln(' Convergence not obtained after 50 iterations')
            END
        END;
    END.

```

يحتوى البرنامج على عدد من مكونات التحكم . ويتم التحكم بصفة خاصة فى التكرار الفعلى بواسطة مكون WHILE - DO . وكان من الممكن استخدام مكون REPEAT - UNTIL بدلا منه أيضا . وبالإضافة إلى ذلك ... يوجد العديد من مكونات IF - THEN - ELSE , IF - THEN داخل البرنامج . لاحظ استخدام مكونات IF - THEN - ELSE المتداخلة ناحية منتصف البرنامج .

لاحظ العبارة التالية :

```
root := exp(0.2*ln(test));
```

والموجودة بقرب منتصف البرنامج . هذه العبارة مطلوبة لترفع قيمة test إلى القوة 0.2 ( تذكر أن البسكال لا يحتوى على عملية أسية ) . وعلى هذا ... فإننا نحصل على اللوغاريتم الطبيعي log للمتغير test ، ونضربه فى 0.2 ، ثم نرفع e للقوة المساوية لنتيجة حاصل الضرب .

والمخرجات الناتجة من اختيار قيمة تقديرية أولية تساوى 1 للمتغير x موشحة أدناه . ( وإجابات المستفيد موضوع تحتها خط ) . لاحظ أن الحسابات التى حدث لها تقارب إلى الحل بأن x مساوية 1.35196 ، وذلك بعد 16 تكرارا ، وتوضح المخرجات المطبوعة أن القيم المتتالية للمتغير x أصبحت متقاربة أكثر ؛ مما أدى إلى الحل النهائى .

```
Initial guess= 1
Iteration number 1  x= 1.47577
Iteration number 2  x= 1.28225
Iteration number 3  x= 1.38344
Iteration number 4  x= 1.33613
Iteration number 5  x= 1.35952
Iteration number 6  x= 1.34826
Iteration number 7  x= 1.35375
Iteration number 8  x= 1.35109
Iteration number 9  x= 1.35238
Iteration number 10 x= 1.35175
Iteration number 11 x= 1.35206
Iteration number 12 x= 1.35191
Iteration number 13 x= 1.35198
Iteration number 14 x= 1.35195
Iteration number 15 x= 1.35196
Iteration number 16 x= 1.35196
Root= 1.35196 . No. of iterations= 16
```

افرض الآن أن القيمة 10 تم اختيارها للمتغير x كتقدير ابتدائى . ينتج عن هذا الاختيار قيمة سالبة للمتغير test فى أول محاولة . وعلى هذا ... تظهر المخرجات على النحو التالى :

```
Initial guess= 10
Numbers out of range - try again using another initial guess
```

ومن الممتع رؤية ماذا يحدث عندما يكون التقدير الابتدائى مساويا 1 ، مع تقليل أقصى عدد لمرات التكرار من 50 إلى 10 مرات . ونشجع القارئ بأن يحاول عمل ذلك ، مع ملاحظة النتائج .

## 7. THE CASE STRUCTURE : ٧ - مكون الحالة :

مكون الحالة هو مكون تحكم شرطى يسمح باختيار مجموعة عبارات محددة من عدة مجموعات متاحة . ويعتمد الاختيار على القيمة الحالية لتعبير يشار إليه بأنه القائم بالاختيار selector .

والصيغة العامة لمكون الحالة هي :

```
CASE expression OF
  case label list 1 : statement 1;
  case label list 2 : statement 2;
  .
  .
  .
  case label list n : statement n
END
```

يمكن أن يكون التعبير بسيطاً ، على ألا يكون حقيقياً . وعادة ما يأخذ صورة متغير فردي بسيط .

وكل عنوان label من عناوين الحالة يمثل إحدى القيم المسموح بها للتعبير . وعلى هذا ... فإذا كان التعبير من النوع الصحيح ، فإن عناوين labels الحالة تمثل قيماً صحيحة تقع داخل مدى مسموح به . ولاتحتاج عناوين الحالة لأن تكون مرتبة ترتيباً معيناً ، بالرغم من أن كل عنوان من هذه العناوين يجب أن يكون فردياً . والأكثر من هذا ... يمكن أن يظهر كل عنوان في قائمة list واحدة فقط . ( ملاحظة : يشار إلى هذه العناوين بأنها عناوين الحالة case labels ، وذلك لتمييزها عن أنواع العناوين الأخرى ، والتي سوف تناقش في القسم التالي ) .

ويمكن أن تكون العبارات بسيطة أو مرتبة . ومن الشائع جداً استخدام العبارات المركبة . ويسمح أيضاً باستخدام العبارة الخالية لتحديد أنه لن تتخذ أى إجراءات لقيم معينة من قيم القائمة بالاختيار selector . ولاتحتاج العبارات لأن تكون فردية ( أى أنه يمكن استخدام نفس العبارة مع قائمتين أو أكثر من عناوين الحالة ) .

وتنفذ العبارة إذا متوافقت عناوين الحالة المناظرة لها فقط مع القيمة الحالية للتعبير . وعلى هذا ... فإن القيمة الحالية للتعبير تحدد العبارة التي سوف تنفذ . فإذا لم تتوافق القيمة الحالية للتعبير مع أى عناوين ، فلا يكون الإجراء معروفاً . ( بعض صيغ البسكال تحتوى على جزء " وإلا otherwise " والذي يحدد الإجراء الذى يتخذ إذا لم تتوافق قيمة القائمة بالاختيار selector مع أى من العناوين ) .

#### مثال (٦-١٥)

فيما يلي مكون حالة تقليدي . وفى هذه الحالة ... فإن choice من النوع الحرفى :

```
CASE choice OF
  'R' : writeln(' RED ');
  'W' : writeln(' WHITE ');
  'B' : writeln(' BLUE ');
END;
```

يطبع RED إذا ما تحددت R كقيمة للمتغير choice . وتُظهر WHITE إذا ما حددت W كقيمة للمتغير choice ، كما تكتب BLUE إذا ما تحددت B كقيمة للمتغير choice . ولا ينتج أى مخرجات إذا ما تحدد أى رمز آخر للمتغير choice غير الحروف R أو W أو B .

#### مثال (٦-١٦)

فيما يلي مثلاً تقليدياً آخر لمكون الحالة :

```
CASE trunc(x/10) OF
  1 : y := y+5;
  3,5 : y := y-2;
  6 : y := 2*(y+1);
  2,4 : ;
  9 : y := 0
END;
```

فى هذا المثال x ، y من النوع الحقيقى ، ويحذف الكسر العشري من قيمة  $x/10$  . ويستخدم الجزء الصحيح كقائم بالاختيار . فإذا كان الجزء الصحيح مساوياً 1 ، فسوف تزداد قيمة y بمقدار 5 . وبالمثل فإن قيمة y تقل بمقدار 2 إذا ما كان الجزء الصحيح مساوياً 3 أو 5 . أما إذا كان الجزء الصحيح مساوياً 6 ، فتتحدد قيمة y بأنها  $2*(y+1)$  .

كما تحدد قيمة  $y$  بأنها 0 إذا ما كان الجزء الصحيح مساويا 9 . وتظل قيمة  $y$  كما هي بدون تغيير إذا ما كانت قيمة الجزء الصحيح 2 أو 4 . ولاتكون الإجراءات معرفة لكل قيم القائم بالاختيار الأخرى ، أى أنه لاتتخذ أية إجراءات أخرى ( ويمكن إنتاج رسالة خطأ في هذه الحالة مثلا ) .

ومن الناحية العملية ... يمكن التفكير في مكون الحالة كبديل لاستخدام مكونات IF - THEN - ELSE . وعلى أية حال ... فإن هذا المكون يمكنه أن يحل محل مكونات IF - THEN - ELSE التى تختبر المعادلات فقط . وفى هذه الحالة عادة مايكون استخدام مكون الحالة مريحا .

### مثال (٦-١٧)

حساب الاستهلاك . دعنا نأخذ في الاعتبار كيف يحسب الاستهلاك السنوى لبعض العناصر المستهلكة ( مثل المبنى أو المكنة وغيرهما ) . وهناك ثلاث طرق شائعة الاستخدام لحساب الاستهلاك ، وهى طريقة الخط المستقيم straight-line ، وطريقة القسط المتناقص double declining balance وطريقة مجموع الأرقام الدالة على السنوات sum-of-the-year's digits . ونريد كتابة برنامج بسكال يسمح لنا باختيار أى طريقة من هذه الطرق لكل مجموعة من الحسابات .

وتبدأ الحسابات بقراءة القيمة الأصلية ( التى لم يحدث لها استهلاك ، وتعرف بأنها القيمة الابتدائية أيضا ) للعنصر والعمر الافتراضى للعنصر ( أى عدد السنوات المتوقع أن يستهلك خلالها العنصر ) ورقم يحدد أى الطرق الثلاث لحساب الاستهلاك هى المستخدمة . ويعد ذلك يحسب الاستهلاك السنوى والقيمة المتبقية ( غير المستهلكة ) للعنصر وتكتب ، وذلك لكل سنة .

طريقة الخط المستقيم هى أسهل الطرق الثلاث . وفى هذه الطريقة تقسم القيمة الأصلية للعنصر على عمر العنصر الافتراضى ( إجمالى عدد السنين ) . وخارج القسمة هو الكمية المستهلكة فى كل سنة ، فإذا ما كان أحد العناصر له قيمة أصلية 8000 دولار وعمره الافتراضى عشر سنوات ، فإن قيمة الاستهلاك السنوى تكون 800 دولار ، بحيث أن العنصر تقل قيمته بمقدار 800 دولار سنويا . لاحظ أن الاستهلاك السنوى ثابت فى كل السنوات .

وعند استخدام طريقة القسط المتناقص ، فإن قيمة العنصر تقل بنسبة سنوية ثابتة . ( وعلى هذا ... فإن القيمة الفعلية للاستهلاك بالدولار تتغير من سنة لأخرى ) ، والحصول على معامل الاستهلاك ، فإننا نقسم 2 على العمر الافتراضى للعنصر ، ثم يضرب هذا المعامل فى قيمة العنصر فى بداية كل سنة ( وليس فى القيمة الأصلية أو القيمة الابتدائية للعنصر ) ، وذلك للحصول على قيمة الاستهلاك السنوى .

افرض على سبيل المثال أننا نريد حساب استهلاك عنصر قيمته الأصلية 8000 دولار على مدار 10 سنوات باستخدام طريقة القسط المتناقص . معامل الاستهلاك يكون  $0.20 = 2/10$  ، وعلى ذلك ... فإن قيمة الاستهلاك فى السنة الأولى هى  $8000 \times 0.2 = 1600$  دولار . وقيمة الاستهلاك فى السنة الثانية هى  $8000 - 1600 = 6400$  .  $6400 \times 0.2 = 1280$  (1600 دولار . وقيمة الاستهلاك فى السنة الثالثة هى  $5120 \times 0.2 = 1024$  دولاراً ، وهكذا .

وفى طريقة مجموع الأرقام الدالة على السنوات تقل قيمة العنصر بنسبة تختلف من سنة لأخرى . ويكون معامل الاستهلاك كسرا مقامه عبارة عن مجموع الأرقام من 1 إلى  $n$  ، حيث  $n$  هى العمر الافتراضى للعنصر ( فإذا كان العمر الافتراضى 10 ، فإن المقام يكون  $1+2+3+...+10 = 55$  ) . وبالنسبة للسنة الأولى يكون البسط  $n$  . وبالنسبة للسنة الثانية يصبح البسط  $n-1$  . وفى السنة الثالثة  $n-2$  وهكذا . ويتم الحصول على قيمة الاستهلاك السنوى بضرب معامل الاستهلاك فى القيمة الأصلية للعنصر .

ولمعرفة كيف تعمل طريقة جمع الأرقام الدالة على السنين ، فإننا نحسب استهلاك نفس العنصر الذى قيمته الابتدائية 8000 دولار ، وعمره الافتراضى 10 سنوات ، قيمة الاستهلاك للسنة الأولى هى  $1454.55 = (10/55) \times 8000$  دولار ، والسنة الثانية  $1309.09 = (9/55) \times 8000$  ، وهكذا .



دعنا الآن نعرف الرموز التالية :

val = قيمة العنصر  
tag = القيمة الأصلية للعنصر ( أى القيمة الأصلية للمتغير val )  
deprec = قيمة الاستهلاك السنوى  
n = العمر الافتراضى للعنصر  
year = عداد تتراوح قيمة من 1 إلى n  
choice = رقم صحيح يحدد الطريقة المستخدمة لحساب الاستهلاك  
flag = متغير بوليان يحدد ما إذا كانت مجموعة الحسابات تنفذ أم لا

ويتبع برنامج البسكال التخطيط التالى :

(١) توضيح المتغيرات .

(٢) تحديد قيمة صحيح للمتغير flag .

(٣) تكرار الخطوات التالية ، حتى تصبح قيمة flag خطأ :

(أ) قراءة قيمة للمتغير choice ( أما 1 أو 2 أو 3 أو 4 لتحديد نوع الحسابات التى تستخدم ) .

(ب) إذا لم تكن قيمة choice هي 4 ، تقرأ قيم val ، n ( إذا كانت قيمة choice هي 4 ، فإن هذا يحدد نهاية التنفيذ ) .

(ج) إذا كانت قيمة choice هي 1 أو 2 أو 3 ، يحسب الاستهلاك السنوى والقيمة الجديدة للعنصر طبقا للطريقة المناسبة ، والتى تحدد طبقا لقيمة choice ، وتطبع النتائج سنة بسنة . أولا ( إذا كانت قيمة هي 4 ) : توضع قيمة خطأ للمتغير flag ، وتكتب رسالة ، وتفصل الحسابات .

والآن دعنا نعبر عن هذا التخطيط بالشفرة الشبيهة .

```
PROGRAM depreciation1(input,output);
(* variable declarations *)
BEGIN
(* initialize flag *)
REPEAT
(* read choice *)
IF choice <> 4 THEN (* read val and n *)
CASE choice OF
1 : BEGIN
(* write heading: Straight-Line Method *)
deprec := val/n;
FOR year := 1 TO n DO
BEGIN
val := val-deprec;
(* write year, deprec, val *)
END
END
```

(تكملة البرنامج فى الصفحة التالية)

```

END;
2 : BEGIN

    (* write heading: Double Declining Balance Method *)

    FOR year := 1 TO n DO
        BEGIN
            deprec := 2*val/n;
            val := val-deprec;

            (* write year, deprec, val *)

        END
    END;
3 : BEGIN

    (* write heading: Sum-of-the-Years'-Digits Method *)

    tag := val;
    FOR year := 1 TO n DO
        BEGIN
            deprec := (n-year+1)*tag/(n*(n+1)/2);
            val := val-deprec;

            (* write year, deprec, val *)

        END
    END;
4 : BEGIN

    (* write message *)

    flag := false;
    END
END
UNTIL flag = false
END.

```

ومعظم الشفرة الشبيهة مباشرة ، بالرغم من وجود بعض التعليقات خلالها . أولا نرى أن مكون REPEAT-UNTIL مستخدم لتكرار مجموعة من الحسابات ، وداخل هذا المكون استخدام مكون الحالة CASE لاختيار طريقة استهلاك محددة . وكل طريقة استهلاك تستخدم مكون FOR لعمل دورة خلال عمر العنصر الافتراضى ، والمقدر بانه ( n - year ) من السنوات .

ويبدو حساب الاستهلاك باستخدام طريقة مجموعة الأرقام الدالة على السنين بهما . والاصطلاح ( n - year + 1 ) يحتاج بصفة خاصة إلى توضيح . وتستخدم هذه الكمية فى العد التنازلى من n وحتى 1 مع تقدم السنين للأمام ( من 1 إلى n ) . وهذه القيم المتناقصة مطلوبة فى طريقة مجموعة الأرقام الدالة على السنين . وقد كان يمكننا بالطبع أن نعد دورة عد تنازلية بدلا من ذلك ( أى يمكن استخدام ( FOR year := n DOWNT0 1 DO ) ، إلا أن ذلك يتطلب كتابة دورة عد تصاعديّة مناظرة لإخراج الاستهلاك السنوى لجميع السنين فى الترتيب المناسب .

وعند هذه النقطة لا يكون من الصعب كتابة برنامج يسكال كاملا على النحو التالي :

```

PROGRAM depreciation1(input,output);

(* THIS PROGRAM CALCULATES DEPRECIATION INTERACTIVELY,
   USING ONE OF THREE POSSIBLE METHODS *)

VAR n,choice,year : integer;
    val,deprec,tag : real;
    flag : boolean;

BEGIN (* action statements *)
    flag := true;
    REPEAT
        page;      (* Begin input routine *)
        write(' Method: (1-SL 2-DDB 3-SYD 4-End) ');
        readln(choice);
        IF choice <> 4 THEN
            BEGIN
                write(' Original value: ');
                readln(val);
                write(' Number of years: ');
                readln(n);
                writeln
            END;

        CASE choice OF

            1 : BEGIN (* SL *)
                    writeln(' Straight-Line Method');
                    writeln;
                    deprec := val/n;
                    FOR year := 1 TO n DO
                        BEGIN
                            val := val-deprec;
                            write(' End of Year ',year:2);
                            write(' Depreciation: ',deprec:5:0);
                            writeln(' Current Value: ',val:6:0)
                        END
                    END;

            2 : BEGIN (* DDB *)
                    writeln(' Double Declining Balance Method');
                    writeln;
                    FOR year := 1 TO n DO

                        BEGIN
                            deprec := 2*val/n;
                            val := val-deprec;
                            write(' End of Year ',year:2);
                            write(' Depreciation: ',deprec:5:0);
                            writeln(' Current Value: ',val:6:0)
                        END
                    END;
        END;
    UNTIL flag = false;
END;
```

(تكملة البرنامج في الصفحة التالية)

```

3 : BEGIN (* SYD *)
    writeln(' Sum-of-the-Years''-Digits Method');
    writeln;
    tag := val;
    FOR year := 1 TO n DO
        BEGIN
            deprec := (n-year+1)*tag/(n*(n+1)/2);
            val := val-deprec;
            write(' End of Year ',year:2);
            write(' Depreciation: ',deprec:5:0);
            writeln(' Current Value: ',val:6:0)
        END
    END;

4 : BEGIN (* end computation *)
    writeln(' That''s all, folks!');
    flag := false;
    END
END (* choice *)
UNTIL flag = false
END.

```

وقد صمم البرنامج للعمل المتداخل ، مع وجود ملفقات لبيانات المدخلات المطلوبة . لاحظ أن البرنامج ينتج قائمة بها أربعة اختيارات لحساب الاستهلاك باستخدام إحدى الطرق الثلاث ، أو لإنهاء الحسابات . وسوف يستمر الكمبيوتر في قبول مجموعات جديدة من بيانات المدخلات . وأداء الحسابات المناسبة لكل مجموعة من مجموعة البيانات ، حتى يتم اختيار المقدار 4 من القائمة .

وموضح أدناه بعض المخرجات ، حيث وضع خط تحت استجابات المستفيد . وفي كل حالة يحدث استهلاك لعنصر قيمته الأصلية 8000 دولار ، وذلك على عشر سنوات باستخدام كل طريقة من الطرق الثلاث . وأخيرا تفصل الحسابات كاستجابة لآخر اختيار في القائمة .

Method: (1-SL 2-DDB 3-SYD 4-End) 1  
 Original value: 8000  
 Number of years: 10

#### Straight-Line Method

End of Year	1	Depreciation:	800.	Current Value:	7200.
End of Year	2	Depreciation:	800.	Current Value:	6400.
End of Year	3	Depreciation:	800.	Current Value:	5600.
End of Year	4	Depreciation:	800.	Current Value:	4800.
End of Year	5	Depreciation:	800.	Current Value:	4000.
End of Year	6	Depreciation:	800.	Current Value:	3200.
End of Year	7	Depreciation:	800.	Current Value:	2400.
End of Year	8	Depreciation:	800.	Current Value:	1600.
End of Year	9	Depreciation:	800.	Current Value:	800.
End of Year	10	Depreciation:	800.	Current Value:	0.

Method: (1-SL 2-DDB 3-SYD 4-End) 2  
 Original value: 8000  
 Number of years: 10

#### Double Declining Balance Method

End of Year	1	Depreciation:	1600.	Current Value:	6400.
End of Year	2	Depreciation:	1280.	Current Value:	5120.
End of Year	3	Depreciation:	1024.	Current Value:	4096.

(تكملة البرنامج في الصفحة التالية)

End of Year 4	Depreciation: 819.	Current Value: 3277.
End of Year 5	Depreciation: 655.	Current Value: 2621.
End of Year 6	Depreciation: 524.	Current Value: 2097.
End of Year 7	Depreciation: 419.	Current Value: 1678.
End of Year 8	Depreciation: 336.	Current Value: 1342.
End of Year 9	Depreciation: 268.	Current Value: 1074.
End of Year 10	Depreciation: 215.	Current Value: 859.

Method: (1-SL 2-DDB 3-SYD 4-End) 3

Original value: 8000

Number of years: 10

Sum-of-the-Years'-Digits Method

End of Year 1	Depreciation: 1455.	Current Value: 6545.
End of Year 2	Depreciation: 1309.	Current Value: 5236.
End of Year 3	Depreciation: 1164.	Current Value: 4073.
End of Year 4	Depreciation: 1018.	Current Value: 3055.
End of Year 5	Depreciation: 873.	Current Value: 2182.
End of Year 6	Depreciation: 727.	Current Value: 1455.
End of Year 7	Depreciation: 582.	Current Value: 873.
End of Year 8	Depreciation: 436.	Current Value: 436.
End of Year 9	Depreciation: 291.	Current Value: 145.
End of Year 10	Depreciation: 145.	Current Value: 0.

Method: (1-SL 2-DDB 3-SYD 4-End) 4

That's all, folks!

لاحظ أن آخر طريقتين ينتج عنهما استهلاك سنوي كبير خلال السنوات الأولى ، واستهلاك سنوي صغير جدا في آخر سنوات العمر الافتراضي للعنصر ، كما نرى أيضا أن قيمة العنصر تصبح صفرا في نهاية عمره الافتراضي عند استخدام كل من الطريقة الأولى والطريقة الثالثة ، إلا أنه عند استخدام الطريقة الثانية تبقى قيمة صغيرة غير مستهلكة عند انتهاء عمر العنصر الافتراضي .

## 8. THE GOTO STATEMENT

٨ - عبارة اذهب إلى :

عبارة اذهب إلى GOTO هي عبارة بسيطة تستخدم لإبدال تسلسل تنفيذ البرنامج ، وذلك بنقل التحكم ( أي بعمل قفزات ) إلى جزء بعيد من البرنامج . وصيغة عبارة GOTO هي :

GOTO statement label

حيث يمثل عنوان العبارة statement label رقما موجبا ، لا يزيد عن 9999 .

مثال (٦-١٨)

فيما يلي عدة عبارات GOTO تقليدية :

GOTO 100;

IF flag THEN GOTO 9999;

IF sum > 100.0 THEN BEGIN

writeln(sum);

GOTO 35

END;

ويجب توضيح كل عناوين العبارات قبل إمكانية استخدامها داخل البرنامج . ( تذكر أن توضيح العناوين label declaration يجب أن يسبق توضيحات الثوابت والمتغيرات ، كما سبق ذكره في قسم ٥ من الفصل الأول ، وقسم ٢ من الفصل الخامس ) . وتأخذ توضيحات العناوين الصورة التالية :

حيث label 1 statement ، و label 2 statement .. تمثل عناوين فردية العبارات . ( لاحظ التمييز بين عناوين العبارات statement labels وعناوين الحالة Case labels التي سبق ذكرها في القسم السابق . لاحظ بصفة خاصة أن عناوين العبارات مقيدة ، بحيث أنها يجب أن تكون أرقاما موجبة ، بينما عناوين الحالة ليست مقيدة . كما أن عناوين العبارات يجب توضيحها قبل أن يمكن استخدامها ، بينما لا يتم أى توضيح لعناوين الحالة ) .

#### مثال (٦-١٩)

فيما يلي جزءاً من برنامج بسكال يحتوى على توضيح عناوين :

```
PROGRAM sample(input,output);
LABEL 100,200,300;
CONST factor = 0.5;
VAR a,b,c : real;
```

لاحظ أن توضيح العناوين يلي عبارة PROGRAM ، ويسبق تعريف الثوابت ، طبقاً لما هو مطلوب .

وتكتب العبارات التي لها عناوين على النحو التالي :

*statement label : statement*

وقد تسبق مثل هذه العبارات عبارات GOTO المناظرة لها ، أو تليها .

#### مثال (٦-٢٠)

```
فيما يلي عدة عبارات بسكال . كل منها يحتوى على عنوان لعبارة
10 : readln(a,b,c);

20 : FOR count := 1 TO n DO
      BEGIN
          .
          .
          .
      END;

99 : BEGIN
      writeln(sum);
      IF flag THEN writeln(x,y,z);
      writeln
      END;
```

ويتسبب تنفيذ عبارة GOTO في نقل التحكم إلى العبارات المناظرة لهذه العناوين . وعلى هذا ... فتكون العبارة التي لها العنوان هي التي تنفذ بعد GOTO .

ويمكن أن يحتوى البرنامج على عدة عبارات GOTO تنقل التحكم إلى نفس الموقع ( إلى نفس العبارة البعيدة ) داخل البرنامج . وعلى أية حال ... يجب أن يكون لكل عبارة معنونة عنوان خاص بها ( أى أنه لا يمكن أن تأخذ عبارتان مختلفتان نفس العنوان ) .

## مثال (٦-٢١)

فيما يلي جزءاً من برنامج بسكال :

```
PROGRAM sample(input,output);
LABEL 10,20;
CONST . . .;
VAR . . .;
BEGIN
.
.
.
10 : readln(a,b,c);
.
.
.
IF a <= 0 THEN GOTO 20;
.
.
.
GOTO 10;
.
.
.
IF flag THEN GOTO 20;
.
.
.
20 : writeln(x,y,z);
.
.
.
END.
```

لاحظ أنه يمكن نقل التحكم إلى العبارة رقم 20 من موقعين مختلفين داخل البرنامج . وكل من عملتي النقل شرطية ، أى أنها تعتمد على صحة أحد تعبيرات بوليان .

ويجب تجنب مثل هذا النوع من العبارات في كتابة برنامج البسكال بصفة عامة . وفى واقع الأمر ... فإن العديد من سمات البسكال صممت خصيصاً لمنع الحاجة لهذا المنطق في البرنامج . ( سوف يذكر فيما بعد المزيد عن ذلك ) .

ويجب الحرص عند استخدام عبارة GOTO مع العبارات المركبة ، فيمكن نقل التحكم خارج out of أو داخل Within العبارة المركبة ، كما يمكن نقل التحكم إلى بداية beginning العبارة المركبة . ولا يمكن على أية حال نقل التحكم إلى داخل into عبارة مركبة . والأكثر من هذا ... إذا ما نقل التحكم داخلياً إلى END الخاصة بعبارة مركبة ( أى إذا ما وضع عنوان للكلمة الرئيسية END ) ، فيجب أن يسبق العنوان عبارة فارغة . ويتحقق ذلك عن طريق وضع فاصلة منقولة بعد العبارة التى تسبق END .

وهذه القيود موضحة فى الأمثلة التالية :

## مثال (٦-٢٢)

فيما يلي مثالا لنقل مسموح به للتحكم خارج عبارة مركبة .

```

BEGIN
.
.
.
IF flag THEN GOTO 50;
.
.
.
END;
.
.
.
50 : writeln(answer);

```

#### مثال (٢٣-٦)

فيما يلي مثالا لنقل مسموح به للتحكم داخل عبارة مركبة .

```

BEGIN
.
.
.
IF flag THEN GOTO 20;
.
.
.
20 : writeln(answer);
.
.
.
END;

```

#### مثال (٢٤-٦)

اعتبر الآن نقل التحكم التالي إلى END الخاصة بعبارة مركبة .

```

BEGIN
.
.
.
IF flag THEN GOTO 30;
.
.
.
writeln(answer);
30 : END;

```

لاحظ الفاصلة المنقوطة الموجودة بعد عبارة Writeln . ينتج عن ذلك عبارة فارغة بين Writeln و END كما هو مطلوب تماما .



### مثال (٦-٢٥)

اعتبر أخيراً المحاولة غير المسموح بها illegal التالية لنقل التحكم إلى داخل عبارة مركبة .

```
IF flag.THEN GOTO 40;
.
.
.
BEGIN
.
.
.
40 : writeln(answer);
.
.
.
END;
```

إذا ما احتوى أحد البرامج مثل هذا التكوين ، فينتج عن ذلك خطأ تكويني أثناء عملية الترجمة .

وهناك قيود شبيهة على استخدام عبارة GOTO مع مكونات التحكم الأخرى ، فيمكن نقل التحكم داخل مكون التحكم أو خارجه ، ولا يمكن نقله إلى داخله . وأكثر من ذلك ... لا تقبل بعض مترجمات البسكال وجود عبارات معنونة داخل مكونات WHILE و FOR و IF . وعلى هذا ... فمن المفضل تجنب استخدام عبارات GOTO في مثل هذه الحالات .

كما أنه هناك قيود أيضاً على استخدام عبارة GOTO التي تنتمي إلى مدى البرنامج program scope . وسوف يناقش هذا الأمر في الفصل القادم . وفي كلمات عامة سوف نرى أنه يمكن إجراء تداخل للمجموعات blocks كما يحدث ذلك في مكونات التحكم الموجودة في البرنامج . وفي مثل هذه الحالات يمكن نقل التحكم داخل المجموعة block ، ومن خارج المجموعة إلى مجموعة مشمولة enclosing ، ولا يمكن نقل التحكم إلى داخل مجموعة متوازنة .

وأخيراً يجب تمييز أن استخدام عبارات GOTO غير مرغوب فيه في البسكال ، حيث إنه يبدل من المسار التتابعي للمنطق ، ويقلل من الوضوح ، وهذا عكس ما تتميز به هذه اللغة . وفي واقع الأمر هناك بعض علماء الكمبيوتر الذين يضعون حظراً كاملاً على استخدام عبارات GOTO ، بالرغم من أن هذا يعتبر تطرفاً بعض الشيء . وهناك اتفاق شبه عام على أنه يجب تجنب استخدام GOTO بقدر الإمكان .

وعملياً يمكن أن توجد عبارة GOTO داخل مكون IF ، وذلك لنقل التحكم ( شرطياً ) إلى نهاية البرنامج . ( تفضل مكونات WHILE-DO و REPEAT - UNTIL في المواقف التي تتطلب نقل التحكم إلى بداية البرنامج ) . وعادة ما تحدث هذه التنقلات بالاتصال مع بعض أنواع الاستراتيجيات العامة global ( أى القفز إلى نهاية البرنامج إذا ما كانت إحدى قيم المدخلات سالبة ) . وفي الناحية الأخرى يجب ألا يحتوى البرنامج تحت أى ظروف على عدد كبير من القفزات المحلية localized ، فمثل هذه البرامج تتنافى مع تصميم لغة البسكال الذي يهدف لوضوح منطق البرنامج .

### مثال (٦-٢٦)

حساب قيمة متوسط قائمة من الأعداد . دعنا نعتبر مرة أخرى مشكلة حساب متوسط قائمة من الأعداد ، كما في مثال ٦ - ٤ ، ومثال ٦ - ٥ ، ومثال ٦ - ٨ . افترض الآن أن طول القائمة غير معروف مسبقاً وبدلاً من ذلك علينا أن نستمر في قراءة وجمع كميات مدخلات متتالية ، حتى نقرأ قيمة سالبة في الكمبيوتر . وهذه القيمة السالبة تفسر بأنها معيار للتوقف ، ولا تدخل في حساب المتوسط .

وتجرى الحسابات على النحو التالي :

- (١) تحديد قيمة 0 للمتغير الصحيح count .
  - (٢) تحديد قيمة 0 للمتغير الحقيقي sum .
  - (٣) تحديد قيمة صحيح للمتغير البولياني flag .
  - (٤) تكرار أداء الخطوات التالية ، طالما أن قيمة flag صحيحة true .
  - (أ) قراءة أحد الأعداد من القائمة ( كل عدد ممثل بمتغير حقيقي x ) .
  - (ب) إذا كانت قيمة x سالبة يتجه التحكم GOTO إلى الخطوات رقم 5 ، وإلا فتضاف القيمة x إلى sum ، وتزداد قيمة count بمقدار 1 .
  - (٥) قسمة sum على count للحصول على قيمة المتوسط .
  - (٦) طباعة أو كتابة قيم count أو average .
- وفيما يلي برنامج البسكال المناظر لذلك .

```
PROGRAM average5(input,output);

(* THIS PROGRAM CALCULATES THE AVERAGE OF A LIST OF NUMBERS .
   USING THE WHILE - DO STRUCTURE AND A GOTO STATEMENT *)

LABEL 10;

VAR count : integer;
    x,sum,average : real;
    flag : boolean;

BEGIN (* action statements *)
    count := 0;
    sum := 0;
    flag := true;
    WHILE flag DO
        BEGIN
            readln(x);
            IF x < 0 THEN GOTO 10;
            sum := sum+x;
            count := count+1
        END;
    10 : average := sum/count;
        writeln(' The average of ',count:5,' numbers is ',average)
    END.
```

لاحظ أن قيمة flag تظل صحيحة true خلال هذا البرنامج . وعلى هذا ... فيستمر التكرار دون توقف حتى تظهر قيمة سالبة للمتغير x ، ويتم إدخالها داخل الكمبيوتر . وبمجرد اكتشاف قيمة سالبة للمتغير x ، ينقل التحكم خارج الدورة إلى العبارة رقم 10 . عند ذلك تسحب قيمة المتوسط ، وتكتب كمخرجات .

ويجب أن يحاول القارئ أن يعد هذا البرنامج بنفسه بدون استخدام عبارة GOTO ( انظر مشكلة ٤٥ في نهاية هذا الفصل ) . ويمكن كتابة مثل هذا البرنامج ، إلا أنه أكثر إرباكا عن البرنامج المستخدم لعبارة GOTO .

## Review Questions

## أسئلة للمراجعة :

- (١) ماذا يعنى التكرار repetition ؟
- (٢) ماهى الدورة الشرطية ؟ وماهى الدورة غير الشرطية ؟ ماهى الاختلافات بين الدورة الشرطية ، «الدورة غير الشرطية» ؟
- (٣) ماذا يعنى التنفيذ الشرطى ؟
- (٤) ماذا يعنى الاختيار ؟
- (٥) ماهى القيم التى يمكن أن تمثل بواسطة تعبير بوليان ؟
- (٦) لخص القواعد المصاحبة لتعبيرات بوليان .
- (٧) لخص الفروق بين المؤثرات العلاقية والمؤثرات المنطقية . مانوع العناصر المستخدمة مع كل من هذين النوعين من المؤثرات ؟
- (٨) ماهو الفرق بين العبارات البسيطة والعبارات المركبة ؟
- (٩) اذكر ثلاثة أنواع من العبارات البسيطة .
- (١٠) لخص القواعد التكوينية التى تصاحب العبارات المركبة .
- (١١) ماهو الغرض من مكون WHILE-DO ؟ ومتى يتم تقويم تعبير بوليان ؟ ماهو أقل عدد من المرات ينفذ فيه مكون WHILE-DO ؟
- (١٢) كيف ينتهى تنفيذ مكون WHILE-DO ؟
- (١٣) لخص القواعد التكوينية المصاحبة لمكون WHILE-DO .
- (١٤) ماهو الغرض من مكون REPEAT-UNTIL ؟ وكيف يختلف عن مكون WHILE-DO ؟
- (١٥) ماهو أقل عدد من المرات ينفذ فيه مكون REPEAT-UNTIL ؟ قارن ذلك مع مكون WHILE-DO ، ووضح سبب هذه الاختلافات .
- (١٦) لخص القواعد التكوينية المصاحبة لمكون REPEAT-UNTIL . قارن ذلك بمكون WHILE-DO .
- (١٧) ماهو الغرض من مكون FOR ؟ وكيف يختلف هذا المكون عن مكون WHILE-DO وعن مكون REPEAT-UNTIL ؟
- (١٨) كم عدد مرات تنفيذ مكون FOR ؟ قارن ذلك بمكون WHILE-DO ، ومكون REPEAT-UNTIL .
- (١٩) ماهو الغرض من متغير التحكم الموجود فى مكون FOR ؟ وما نوع المتغير الذى يمكن استخدامه ؟
- (٢٠) صف الصيغتين المختلفتين لمكون FOR . ماهو الغرض من كل منهما ؟
- (٢١) لخص القواعد التكوينية المصاحبة لمكون FOR .

- (٢٢) ماهى القواعد المستخدمة فى عمل مكونات تحكم متداخلة ؟ هل يمكن أن يحتوى أحد أنواع مكونات التحكم نوعا آخر ؟
- (٢٣) ماهو الغرض من مكون IF ؟ وكيف يختلف هذا المكون عن مكون WHILE-DO ، ومكون REPEAT-UNTIL ومكون FOR ؟
- (٢٤) صف صيغتين مختلفتين لمكون IF . وكيف تختلف هاتان الصيغتان عن بعضهما ؟
- (٢٥) لخص القواعد التكوينية المصاحبة لمكون IF.
- (٢٦) كيف تفسر مكونات IF المتداخلة ؟ وبصفة خاصة كيف يفسر المكون التالى :
- (٢٧) ماهو الغرض من مكون CASE ؟ وكيف يختلف هذا المكون عن المكونات الأخرى المذكورة فى هذا الفصل ؟
- (٢٨) ماهو القائم بالاختيار ؟ وماهو الغرض الذى يوجد من أجله ؟
- (٢٩) ماهى القيم التى يمكن تحديدها لعناوين الحالة ؟
- (٣٠) ماذا يحدث عندما تكون قيمة القائم بالاختيار هى نفسها قيمة أحد العناوين ؟ وماذا يحدث عندما لا تكون قيمة القائم بالاختيار هى نفسها قيمة أى عنوان من عناوين الحالة ؟
- (٣١) لخص القواعد التكوينية المصاحبة لتكوين CASE . هل يمكن أن يصاحب عناوين عبارة واحدة ؟
- (٣٢) قارن مكون CASE مع استخدام مكونات IF - THEN - ELSE المتداخلة . أى منها أكثر إقناعا ؟
- (٣٣) ماهو الغرض من عبارة GOTO ؟ وهل هى عبارة بسيطة أم عبارة مركبة ؟
- (٣٤) لخص القواعد التكوينية التى تصاحب عناوين العبارات . ماهى القيم التى يمكن استخدامها كمناوين للعبارات ؟ وكيف يمكن توضيحها ؟ كيف يصاحب عنوان العبارة عبارة بعيدة ؟
- (٣٥) قارن التكوين المصاحب لعناوين العبارات ، مع التكوين المصاحب لعبارات الحالة . لاحظ الفروق .
- (٣٦) صف القيود التى تقع على استخدام عبارة GOTO مع استخدام عبارات مركبة . وهل تطبق نفس القيود عند استخدام عبارة GOTO مع مكونات تحكم أخرى ؟
- (٣٧) لماذا لايشجع استخدام عبارات GOTO فى البسكال ؟ وتحت أى ظروف يمكن أن تكون عبارات GOTO مفيدة ؟ مانوع الاستخدام الذى يجب تجنبه ؟ ولماذا ؟ ناقش ذلك بالتفصيل .

## Solved Problems

## مشاكل محلولة :

(٣٨) حدد قيمة كل تعبير بولياني من التعبيرات التالية ، مستخدما المعرفات المعرفة أدناه :

```
CONST a = 50;
      t = 0.02;
      x = 'P';
```

Boolean Expression	Value
$a < 12$	false
$\text{abs}(t) = a/2500$	appears true, but probably false because of numerical inaccuracies
$x \neq 'p'$	true
$x = a$	error—mixed data types
$(a > 6) \text{ OR } (t \leq 0)$	true
$(a > 6) \text{ AND } (t \leq 0)$	false
$((x = 'P') \text{ AND } (t = 0)) \text{ OR } (x > 'A')$	true
$(t \geq 0) \text{ AND } (\text{NOT } (x \neq 'P'))$	true

(٣٩) كل من هيكل مكون التحكم التالية يوضح طريقة مختلفة لتنفيذ عملية بواره 100 مرة . ( اقرض أن count متغير صحيح ) .

```
(a) count := 1;
    WHILE count <= 100 DO
      BEGIN
        .
        .
        .
        count := count + 1
      END;
```

```
(b) count := 1;
    REPEAT
      .
      .
      .
      count := count + 1
    UNTIL count > 100;
```

```
(c) FOR count := 1 TO 100 DO
    BEGIN
      .
      .
      .
    END;
```

```
(d) FOR count := 100 DOWNT0 1 DO
    BEGIN
      .
      .
      .
    END;
```

(٤٠) كل من التخطيط التالي يوضح طريقة مختلفة لمكونات تحكم متداخلة :

```
(a) VAR i : integer;
    flag : boolean;
    .
    .
    .
    flag := true;
    WHILE flag DO
    BEGIN
        FOR i := 1 TO n DO
        BEGIN
            .
            .
            .
            END;
            .
            .
            .
            IF . . . THEN flag := false
        END;
    END;
```

```
(b) VAR i : integer;
    flag : boolean;
    .
    .
    .
    flag := true;
    REPEAT
    .
    .
    .
    IF . . . THEN FOR i := 1 TO n DO
        BEGIN
            .
            .
            .
            END
        ELSE FOR i := n DOWNT0 1 DO
            BEGIN
                .
                .
                .
            END;
        IF . . . THEN flag := false
    UNTIL NOT flag;
```

```
(c) VAR c : char;
    count : integer;
    .
    .
    .
    IF c <> '+'
```

```

THEN REPEAT
.
.
.
UNTIL c = '*'
ELSE FOR count := 1 TO 3 DO BEGIN
.
.
.
END;
(d) VAR count,start,finish,test,i : integer;
.
.
.
FOR count := start TO finish DO
BEGIN
.
.
.
test := 1;
WHILE test < 10 DO
FOR i := 1 TO test DO
BEGIN
.
.
.
END;
test := test + 1;
.
.
.
END;

```

(٤١) يوضح التخطيط التالي طريقتين مختلفتين لاختيار أحد إجراءات ثلاثة ( افترض أن choice متغير صحيح ) .

```

(a) IF choice = 1 THEN . . .
      ELSE IF choice = 2 THEN . . .
      ELSE . . . ;

(b) CASE choice OF
      1 : . . . ;
      2 : . . . ;
      3 : . . .
      END;

```

لاحظ أن استخدام مكونات IF - THEN - ELSE المتداخلة كجزء من ( a ) يصبح أكثر أرهاقا كلما ازداد عدد الاختيارات .

## Supplementary Problems

### مشاكل متكاملة :

(٤٢) حدد قيمة كل تعبير بولياني من التعبيرات التالية ، مستخدما المعرفات أدناه :

```
CONST f = 300;
      p = -0.001;
      q = 0.001;
      c = '5';
```

- |  |  |
|--|--|
| (a) $2 * f \geq 500$                             | (f) $(p = \text{abs}(q)) \text{ OR } (c > 4)$              |
| (b) $\text{abs}(p) = \text{abs}(q)$              | (g) $\text{sqr}(p) < \text{sqr}(q)$                        |
| (c) $c = 5$                                      | (h) $(q < 0) \text{ OR } ((f > 0) \text{ AND } (f < 100))$ |
| (d) $p + q > 0$                                  | (i) $\text{NOT } (c < '7')$                                |
| (e) $(\text{abs}(p) = q) \text{ AND } (c > '4')$ |  |

(٤٣) فيما يلي تخطيطا لعدة تكوينات تحكم ، وبعضها مكتوب بطريقة خاطئة . حدد كل الأخطاء :

- |   |  |
|---|--|
| <pre>(a) VAR result : real;       .       .       .       WHILE result &gt; 0 DO       BEGIN       .       .       .       END;</pre> | <pre>(d) VAR x : char;       .       .       .       FOR x := 'z' DOWNT0 'a' DO       BEGIN       .       .       .       END;</pre>   |
| <pre>(b) VAR a,b,c,d : integer;       .       .       .       REPEAT       .       .       .       UNTIL (a+b) &gt; (2*c-d);</pre>    | <pre>(e) VAR a,b,c : integer;       .       .       .       FOR a := b TO c DO       BEGIN       .       .       .       END;</pre>  |
| <pre>(c) VAR x,n1,n2 : real;       .       .       .       FOR x := n1 TO n2 DO       BEGIN       .       .       .       END;</pre>  | <pre>(f) VAR a,b,c,d : integer;       .       .       .       FOR a := b DOWNT0 d DO       BEGIN       .       .       .       FOR c := b TO d DO       BEGIN       .       .       .       END       END;</pre> |



(g) VAR a,b,c,d : integer;

FOR b := a TO c DO  
BEGIN

FOR d := a TO b DO  
BEGIN

END

END;

(h) VAR flag : boolean;  
c : char;

flag := true;  
WHILE flag DO  
BEGIN

REPEAT

read(c);  
IF c = '\*' THEN flag := false;

UNTIL NOT flag

END;

(i) VAR i,j : integer;  
x : real;  
flag : boolean;

IF i < 0 THEN IF j > 10 THEN flag := true  
ELSE flag := false ELSE x := -0.1;

(j) VAR x,y : real;

CASE sqr(x) OF

1.0 : y := x;  
4.0 : y := 2\*x;  
9.0 : y := 3\*x;  
16.0 : y := 4\*x

END;

(k) LABEL 10;

VAR c : char;

read(c);

IF c = '\*' THEN GOTO 10;

WHILE c <> '\*' DO

BEGIN

10 : write(c);

END;

(l) LABEL 1,2,3;

VAR i : integer;

CASE i OF

1 : BEGIN . . . GOTO 3 . . . END;

2 : BEGIN . . . GOTO 3 . . . END;

3 : BEGIN . . . END

END;

## Programming Problems

### مشاكل برمجة :

(٤٤) ترجم ونفذ البرامج المعطاه في الأمثلة ٦-٤ ، و ٦-٥ ، و ٦-٨ ، مستخدما العشرة أرقام التالية :

27.5	87.0
13.4	39.9
53.8	47.7
29.2	8.1
74.5	63.2

(٤٥) عدل البرنامج المعطى في مثال ٦ - ٢٦ ، والذي يحسب متوسط قائمة أعداد غير معرف طولها ، مع حذف عبارة GOTO . اكتب برنامجا جديدا بطريقتين مختلفتين :

(أ) باستخدام مكون WHILE-DO .

(ب) باستخدام مكون REPEAT - UNTIL .

اختبر كل صيغة للبرنامج ، مستخدما بيانات المشكلة السابقة .

(٤٦) أعد كتابة برنامج الاستهلاك المعطى في مثال ٦ - ١٧ باستخدام مكون IF - THEN - ELSE ، بدلا من مكون CASE . اختبر البرنامج مستخدما البيانات المعطاه في مثال ٦ - ١٧ . أى صيغة من هاتين الصيغتين تفضلها ؟ وما سبب ذلك ؟

(٤٧) المعادلة :

$$x^5 + 3x^2 - 10 = 0$$

والمقدمة في مثال ٦ - ١٤ يمكن إعادة ترتيبها لتأخذ الصيغة :

$$x = [(10 - x^5)/3]^{1/2}$$

أعد كتابة برنامج البسكال المقدم في مثال ٦ - ١٤ لاستخدام الصيغة المذكورة أعلاه للمعادلة . نفذ البرنامج ، وقارن النتائج الحسوبة مع النتائج المعطاه في مثال ٦ - ١٤ . لماذا تختلف النتائج ؟ هل تخطئ أجهزة الكمبيوتر ؟

(٤٨) عدل البرنامج المعطى في مثال ٦ - ١٤ ، والذي يحل معادلة جبرية مع إحلال مكون WHILE-DO بمكون REPEAT - UNTIL ، أى المكونين يناسب هذه المشكلة الخاصة أفضل ؟

(٤٩) عدل البرنامج المعطى في مثال ٦ - ١٤ ، والذي يحل معادلة جبرية مع إحلال مكون FOR - TO محل مكون WHILE-DO . قارن استخدام مكون FOR - TO مع مكون WHILE-DO ، مع مكون REPEAT - UNTIL . أى هذه المكونات تفضل ؟ ولماذا ؟

(٥٠) اكتب برنامجا كاملا بلغة البسكال لكل مشكلة من المشاكل التالية . استخدام مكون التحكم الأكثر ملائمة لكل مشكلة . ابدأ بتخطيط تفصيلي ثم أعد كتابة هذا التخطيط بالشفرة الشبيهة إذا لم يكن التحويل المباشر إلى برنامج البسكال غير واضح . تأكد من استخدام نظام برمجة جيد ( مع وجود تعليقات وترجيحات وخلافه ) .

(أ) احسب المتوسط المرجح weighted average لقائمة من n عدد مستخدما العلاقة .

$$x_{avg} = f_1x_1 + f_2x_2 + \dots + f_nx_n$$

حيث  $f_1$  و  $f_2$  ، ... هي عوامل ترجيح weighting factors كسرية ، بحيث إن :

$$0 \leq f_i < 1 \quad \text{and} \quad f_1 + f_2 + \dots + f_n = 1$$

اختبر برنامجك بالبيانات التالية :

$i = 1$	$f = 0.06$	$x = 27.5$
2	0.08	13.4
3	0.08	53.8
4	0.10	29.2
5	0.10	74.5
6	0.10	87.0
7	0.12	39.9
8	0.12	47.7
9	0.12	8.1
10	0.12	63.2

(ب) احسب الضرب التراكمي لقائمة من  $n$  عدد . اختبر برنامجك ، مستخدماً مجموعة البيانات التالية :

$$(n = 6): 6.2, 12.3, 5.0, 18.8, 7.1, 12.8$$

(ج) أحسب الوسط الهندسي لقائمة أعداد مستخدماً العلاقة التالية :

$$x_{avg} = [x_1 x_2 x_3 \dots x_n]^{1/n}$$

اختبر برنامجك مستخدماً البيانات المعطاة في الجزء (ب) من المشكلة . قارن النتائج التي تحصل عليها بنتائج المتوسط الحسابي لنفس البيانات . أى متوسط منها هو الأكبر ؟

(د) حدد جذور المعادلة التالية :

$$ax^2 + bx + c = 0$$

مستخدماً الصيغة المعروفة التالية :

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

( انظر مثال ٨ - ٥ ) . خذ في الاعتبار إمكانية أن يكون أحد الثوابت صفراً ، وأن الكمية  $b^2 - 4ac$  تكون أقل من أو تساوى صفراً . اختبر البرنامج مستخدماً مجموعة البيانات التالية :

$a = 2$	$b = 6$	$c = 1$
3	3	0
1	3	1
0	12	-3
3	6	3
2	-4	3

(هـ) أعداد فيبوناكي Fibonacci numbers لها تسلسل يكون فيه كل عدد مساوياً مجموع العددين السابقين له . وفي كلمات أخرى ...

$$F_i = F_{i-1} + F_{i-2}$$

حيث تشير  $F_i$  إلى العدد  $i$  . وأول عددين طبقاً للتعريف يساويان 1 ، أى أن  $F_1 = F_2 = 1$

ملاحظة :

$$F_3 = F_2 + F_1 = 1 + 1 = 2$$

$$F_4 = F_3 + F_2 = 2 + 1 = 3$$

$$F_5 = F_4 + F_3 = 3 + 2 = 5$$

وهكذا :

اكتب برنامجا يحدد أول  $n$  عدد من أعداد فيبوناكي . اختبر البرنامج بقيمة  $n$  مساوية 23

(و) العدد الأولي Prime number هو رقم صحيح موجب ، يمكن قسمته على 1 أو على نفسه فقط بدون باق . احسب واعمل جدولاً من أول  $n$  عدد أولي . ( ملاحظة : العدد  $n$  يكون عدداً أولياً إذا لم يكن خارج قسمة  $n/\sqrt{n}$  )  
(  $n/3, n/2, \dots, n$  ) عبارة عن رقم صحيح . اختبر برنامجك بحساب أول 100 عدد أولي .

(ز) اكتب برنامجا على هيئة حوار يقرأ قيم أرقام صحيحة موجبة ، ويحدد مايلي :

(١) إذا كان الرقم الصحيح عدداً أولياً .

(٢) إذا كان الرقم الصحيح عدداً فيبوناكي .

اكتب برنامجا بطريقة يمكن أن يتكرر بها تنفيذ دورة ، حتى تكتشف قيمة صفر ككمية مدخلة . اختبر البرنامج بعدة قيم صحيحة من اختيارك .

(ح) احسب مجموعة أول  $n$  رقما فرديا ( أى  $1+3+\dots+2*n-1$  ) . اختبر البرنامج بحساب أول 100 رقم فردى ( ملاحظة أن آخر رقم هو 199 ) .

(ط) يمكن حساب جيب  $x$  بالتقريب بتجميع أول  $n$  حد من السلسلة النهائية .

$$\sin x = x - x^3/3! + x^5/5! - x^7/7! + \dots$$

حيث  $n$  يعبر عنه بالتقدير الدائري . اكتب برنامجا بلغة البسكال يقرأ قيمة  $x$  ، ثم يحسب جيبها . اكتب البرنامج بطريقتين مختلفتين .

(١) جمع أول  $n$  حدا ، حيث  $n$  هو رقم صحيح يقرأ داخل الكمبيوتر مع القيمة العددية للزاوية  $x$  .

(٢) استمر في إضافة حدود متتالية في السلسلة ، حتى تصبح قيمة الحد أقل من  $10^{-5}$  ( في قيمتها ) .

اختبر البرنامج لقيم  $x=1$  و  $x=2$  و  $x=3$  . وفي كل حالة اكتب كمخرجات عدد الحدود المستخدمة في الحصول على النتيجة النهائية .

(ي) افرض أن  $P$  من الدولارات استدينت من أحد البنوك ، على أن يعاد دفع  $A$  دولار كل شهر ، حتى يتم دفع القرض كله . وجزء من القسط الشهري يمثل الفائدة محسوبة بأنها  $i\%$  من القيمة الحالية المدان بها . وبقية القسط الشهري تمثل جزءاً مدفوعاً من الدين . اكتب برنامجا بلغة البسكال يحدد المعلومات التالية :

(١) قيمة الفائدة المدفوعة كل شهر .

(٢) قيمة جزء الدين المسدد كل شهر .

(٣) كمية الفائدة المتراكمة التي دفعت للبنك في نهاية كل شهر .

(٤) قيمة القرض التي مازال المدين مدينا بها .

(هـ) عدد الأقساط الشهرية اللازمة لإعادة الدين كاملاً .

(٦) قيمة آخر قسط ( حيث إنها ربما تكون أقل من A ) .

اختبر برنامجك مستخدماً البيانات التالية :

$i = 1\%$  شهرياً .

(ك) حصل مجموعة من الطلبة على الدرجات التالية في ستة امتحانات في لغة البسكال :

Student Number	Exam Scores (percent)					
10001	45	80	80	95	55	75
10002	60	50	70	75	55	80
10003	40	30	10	45	60	55
10004	0	5	5	0	10	5
10005	90	85	100	95	90	90
10006	95	90	80	95	85	80
10007	35	50	55	65	45	70
10008	75	60	75	60	70	80
10009	85	75	60	85	90	100
10010	50	60	50	35	65	70
10011	70	60	75	70	55	75
10012	10	25	35	20	30	10
10013	25	40	65	75	85	95
10014	65	80	70	100	60	95

اكتب برنامجاً تقليدياً بلغة البسكال ، يقبل رقم كل طالب ودرجات الامتحانات كمدخلات ، ويحدد متوسط الدرجة لكل طالب ، ثم يخرج رقم الطالب مع درجات الاختبارات كلها والمتوسط المحسوب . اجعل البرنامج عاماً كلما كان ذلك ممكناً .

(ل) عدل البرنامج المكتوب في (ك) ليسمح بالترجيح غير المتساوي للدرجات الفردية للامتحانات . افرض بصفة خاصة أن كل امتحان من أول أربعة امتحانات يسهم بمقدار 15% من الدرجة النهائية ، وكل من الدرجتين الأخيرتين يسهم بمقدار 20% .

(م) وسع البرنامج الخاص بالجزء (ل) ، بحيث تحسب متوسط درجة الفصل كله بالإضافة إلى متوسطات الطلاب .

(ن) اكتب برنامجاً بلغة البسكال ، يسمح باستخدام الكمبيوتر كحاسبة عادية . اعتبر العمليات الحسابية المعتادة فقط ( الجمع والطرح والضرب والقسمة ) . خذ في الاعتبار ذاكرة يمكنها تخزين رقم واحد .

(س) انتج الهرم التالي من الأرقام ، مستخدماً دورات متداخلة .

```

1
232
34543
4567654
567898765
67890109876
7890123210987
890123454321098
90123456765432109
0123456789876543210

```

( لا تكتب ببساطة 10 سلاسل متعددة الأرقام )

(ع) انتج رسما لدالة .

$$y = e^{-0.1t} \sin(0.5t)$$

عن طريق مطابيع أسطر ، مستخدما نجمة (\*) لكل نقطة تكون الرسم . اجعل الرسم يسرى رأسيا إلى أسفل الصفحة ، مع وجود نقطة واحدة ( نجمة واحدة ) فى السطر . ( ملاحظة : يجب أن يتكون كل سطر من نجمة واحدة ، يسبقها عدد مناسب من الفراغات . حدد موقع النجمة مستخدما دالة round ) .

(ف) اكتب برنامجا بلغة البسكال متداخلا يحول التاريخ الذى يتم إدخاله فى صورة mm-dd-yy ( مثال 69 - 12 - 4 ) إلى رقم يحدد عدد الأيام بعد أول يناير عام ١٩٦٠ . لعمل ذلك استخدم العلاقات التالية :

(١) تاريخ السنة الحالية يمكن تحديده بطريقة تقريبية على النحو التالى :

$$\text{day} := \text{trunc}(30.42 * (\text{mm} - 1)) + \text{dd}$$

(٢) إذا كانت mm = 2 ، أى أن الشهر هو فبراير ، أضف 1 إلى قيمة day .

(٣) إذا كانت mm > 2 ، وكانت mm < 8 ( مارس وأبريل ومايو ويونيو ويوليو ) ، قلل قيمة day بمقدار 1 .

(٤) إذا كان yy MOD = 0 و mm > 2 ( سنة كبيسة ) ؛ أضف 1 إلى قيمة day .

(٥) أضف 1461 إلى day لكل دورة مكونة من 4 سنوات بعد 1-1-60 .

(٦) أضف 365 إلى day لكل سنة كاملة بعد إتمام آخر دورة طولها أقل من 4 سنوات ، ثم أضف 1 ( لأقرب سنة كبيسة ) .

اختبر البرنامج بتاريخ اليوم ، أو بلى تاريخ تختاره .

## الفصل السابع

### الإجراءات والدوال

## Procedures and Functions

لقد رأينا أنه يمكن كتابة برنامج البسكال بسهولة على هيئة أجزاء أو إجراءات modules تسمح بتجزئة المشكلة الكلية إلى سلسلة من المشاكل الجزئية . ويقدم استخدام الأجزاء ميزتين هامتين للغة البسكال . أولا بالنسبة للأنشطة التي يجب تكرارها أكثر من مرة واحدة ، نجد أن استخدام الأجزاء يلغى الحاجة إلى تكرار برمجة هذه الأنشطة بعدد المرات اللازم . وبدلا من ذلك يمكن تعريف جزء البرنامج مرة واحدة ، ثم يتم الاتصال به من أماكن عديدة مختلفة في البرنامج . ويمكن تشغيل مجموعة بيانات مختلفة في كل مرة يتم فيها الاتصال بهذا الجزء . وعلى هذا ... يمكن لاستخدام الأجزاء أن يقلل من طول البرنامج .

كما أن الوضوح في المنطق الناتج من تجزئة البرنامج إلى أجزاء فردية ، حيث يمثل كل جزء شيئا محددا من المشكلة الكلية له دلالة أكبر في واقع الأمر . ومثل هذه البرامج سهلة في كتابتها وفي تصحيحها وفي تكوينها المنطقي عن البرامج التي تفتقد هذا النوع من التكوين . وهذا صحيح بصفة خاصة بالنسبة للبرامج الطويلة والمعقدة . وعلى هذا ... فالعديد من برامج البسكال تستخدم على ذلك الأجزاء ، بالرغم من أنها قد لا تحتوي على تكرارات لنفس الأنشطة . وفي واقع الأمر ، فإن تجزئة البرنامج إلى أجزاء فردية يعتبر بصفة عامة جزءا مهما من البرمجة العملية الجيدة .

وهناك نوعان من أنواع تجزئة البرنامج في البسكال ، وهما الإجراءات procedures ، والدوال functions . وهذان النوعان متشابهان من تكوينهما . ويمكن الاتصال بهما على أية حال بطريقة مختلفة ، وكل منهما يتبادل المعلومات بطريقة مختلفة . وقد ناقشنا بالفعل استخدام إجراءات قياسية ( مثل read و write ) ، ودوال قياسية ( مثل sqr ) في الفصل الثاني . دعنا نعتبر الآن الإجراءات والدوال بتفاصيل أكبر . سوف نرى بصفة خاصة كيف تكتب الإجراءات ، وما هي الاختلافات بين الإجراءات والدوال ، وكيف يمكن استخدام كل منهما استخداما مناسباً . وسوف نعتبر في نهاية الفصل خاصية مهمة للغة البسكال ، تعرف بأنها الإعادة الذاتية recursion ، والتي تجعل من الممكن للإجراء أو للدالة أن تصل إلى نفسها بنجاح .

### 1. PROCEDURES

#### ١ - الإجراءات :

الإجراء procedure هو تكوين برنامج في حد ذاته ، محتوي داخل برنامج البسكال . وفي بعض لغات البرمجة الأخرى يسمى هذا المكون ببرنامج فرعي subroutine أو إجراء فرعي .

ويمكن الإشارة إلى الإجراء بكتابة اسمه ببساطة ، تلووه قائمة اختيارية بالمؤشرات parameters . ويجب أن توضع المؤشرات بين قوسين ، وإذا ما كان هناك أكثر من مؤشر واحد ، فتفصل المؤشرات بواسطة فواصل . ودلائل الإجراء تعرف بأنها موصلات accesses للإجراء ، أو استدعاءات calls للإجراء .

وعند الإشارة إلى إجراء ينتقل التحكم تلقائيا إلى بداية الإجراء . وتنفذ العبارات الإجرائية المحتواة action statements داخل الإجراء عند ذلك ، مع الأخذ في الاعتبار أي توضيحات خاصة تكون فردية بالنسبة للإجراء . وعند الانتهاء من تنفيذ العبارات الإجرائية الموجودة في الإجراء يعود التحكم تلقائيا إلى العبارة التي تلي العبارة التي أشارت إلى الإجراء مباشرة .

### مثال (١-٧)

يقرا أحد برامج البسكال ثلاثة كميات صحيحة ، ثم يحدد أكبر كمية من هذه الكميات . ويحتوى البرنامج على إجراء اسمه maximum ، والذي يحدد أكبر كمية من الثلاث كميات ، ثم يخرجها كنتيجة للبرنامج . والعبارات الإجرائية التى تقرأ الثلاث كميات ، ثم تتصل بالإجراء موضحة أدناه .

```
BEGIN
  readln(a,b,c);
  WHILE a <> 0 DO
    BEGIN maximum; readln(a,b,c) END
  END.
```

وعند ظهور إشارة إلى الإجراء maximum ينتقل التحكم تلقائيا إلى هذا الإجراء ( الإجراء نفسه غير موضح فى المثال ) . ويحدد الإجراء أى من المتغيرات a , b , c هو الأكبر ، ثم يطبع النتائج . بعد ذلك يعود التحكم إلى عبارة readln التالية ( والتى تشير أيضا إلى الإجراء ) .

لاحظ أن البرنامج يقرأ مجموعات متتالية من ثلاث كميات صحيحة ، ويحدد أكبر كمية من هذه الكميات الثلاث فى المجموعة . ويستمر التكرار الذى يتحكم فيها مكون WHILE-DO حتى تتحدد القيمة 0 للمتغير a .

دعنا نعتبر الآن أن الإجراء نفسه مكتوب . وكل إجراء له عنوان خاص به وكتلة block أو مجموعة . ويكتب العنوان على النحو التالى :

PROCEDURE name

أو إذا ما استخدمت مؤشرات رسمية ، فإنه يكون على النحو التالى :

PROCEDURE name(formal parameters)

( وسوف نناقش استخدام المؤشرات الرسمية فى قسم ٧ - ٣ )

وتحتوى الكتلة على جزء توضيح ( وهو جزء محلى للإجراء ) ، ومجموعة من العبارات الإجرائية ، مثل برنامج البسكال الموجود به هذا الإجراء . وعلى هذا ... فيمكن التفكير فى الإجراء على أنه نوع خاص من أنواع برامج البسكال يكون موجودا داخل برنامج البسكال .

### مثال (٢-٧)

فيما يلى إجراء اسمه maximum يحدد أكبر كمية من ثلاث كميات صحيحة هي a , b , c ، ويكتب النتيجة .

```
PROCEDURE maximum;
(* This procedure finds the largest
   of three integer quantities *)
VAR max : integer;
BEGIN
  IF a > b THEN max := a ELSE max := b;
  IF c > max THEN max := c;
  writeln(' The maximum is ',max)
END;
```



أول سطر يحتوى على الكلمة الأساسية PROCEDURE ، وهو عنوان للإجراء . ( لاحظ التشابه مع عنوان البرنامج ) . ويتبع هذا السطر كتلة الإجراء ، والتي تحتوى على توضيح لتغيير واحد وعدة عبارات إجرائية .

لاحظ أن هذا الإجراء يتطلب أن تكون قيم  $a, b, c$  محددة مسبقاً قبل الاتصال بالإجراء . واستخدام الإجراء متوافق مع جزء البرنامج الموضح فى مثال ٧ - ١ .

كل توضيحات الإجراء مثل التوضيحات المذكورة فى المثال السابق يجب أن توجد داخل المجموعات المنادية calling block ( أو داخل مجموعة خارجية تحيط بالمجموعة المنادية ) . ويجب أن تكون توضيحات الإجراء والدالة بصفة خاصة آخر عناصر فى قسم التوضيحات ، وتتبع توضيحات المتغيرات ( انظر قسم ٥ - ١ ، وقسم ٥ - ٢ ) . وبعد توضيح الإجراء داخل المجموعة ، يمكن الاتصال به من أى مكان فى المجموعة .

### مثال (٧-٣)

أكبر عدد من ثلاثة أعداد . فيما يلى برنامجاً كاملاً بلغة البسكال ، يدمج أجزاء البرنامج التى سبق ذكرها فى المثالين السابقين

```
PROGRAM sample(input,output);
(* This program uses a procedure to determine the maximum
   in each set of three integer quantities *)
VAR a,b,c : integer;

PROCEDURE maximum;
(* This procedure finds the largest
   of three integer quantities *)
VAR max : integer;
BEGIN
  IF a > b THEN max := a ELSE max := b;
  IF c > max THEN max := c;
  writeln(' The maximum is ',max)
END;

BEGIN (* main action block *)
  readln(a,b,c);
  WHILE a <> 0 DO
    BEGIN maximum; readln(a,b,c) END
  END.
```

يكبر هذا البرنامج قراءة مجموعات من ثلاثة أعداد صحيحة ، ويحدد أكبر عدد من هذه الأعداد الثلاثة فى كل مجموعة . ويستمر الكمبيوتر فى قراءة وتشغيل مجموعات متتالية من البيانات ، طالما أنه لم يتعرض لقراءة صفر ( حيث إن الصفر يعنى انتهاء هذه المجموعات ) .

لاحظ أن العملية قد تم توضيحها قبل الاتصال بها ، كما أن توضيح الإجراء تلى توضيحات المتغيرات . ( الأسطر الفارغة ظهرت لتحسين قراءة البرنامج فقط ) . ولاحظ أيضاً أن المتغيرات  $a, b, c$  التى سبق تعريفها فى الجزء الرئيسى للبرنامج ، تم تمييزها خلال البرنامج ، بما فيها الإجراء . ومن ناحية أخرى ... تم تعريف max داخل الإجراء فقط . وعلى هذا ... فليس من الممكن استخدام المتغير max داخل الجزء الرئيسى للبرنامج .

والقواعد التى تحكم أدلة الإشارة للإجراء ، تكون عامة فى الواقع عما تحدد الأمثلة سالفة الذكر . وسوف نرى بإيجاز أنه يمكن الاتصال بالإجراء عن طريق الجزء الرئيسى من البرنامج أكثر من مرة واحدة . وأكثر من هذا ... يمكن أن تتصل إجراءات أخرى وبروال أخرى أيضاً بالإجراء .

## 7. SCOPE OF IDENTIFIERS

### ٢ - مدى المعرفة :

يمكن أن توضح الثوابت والمتغيرات التي تظهر داخل العبارات الإجرائية لأداء إجراء معين خارجيا ويكون ذلك داخل مجموعة من مجموعات البرنامج ، والتي تحتوى على توضيح الإجراء نفسه . أو يمكن أن يحدث ذلك محليا ، أى داخل الإجراء نفسه . ويمكن استخدام هذه الثوابت والمتغيرات التي توضح فى المجموعة المحتوية على توضيح الإجراء فى أى مكان داخل هذه المجموعة ، سواء أكان ذلك داخل الإجراء ، أم خارجه . وتعتبر المعرفة التي تعرف بهذه الطريقة أنها شاملة global للإجراء ومن ناحية أخرى ... فإن الثوابت والمتغيرات المحلية local لا تعرف خارج الإجراء . وعلى هذا ... فلا يمكن استخدامها خارجه .

ومدى scope المعرفة يشير إلى المنطقة التي يوضح داخلها المعرفة . وعلى هذا ... يمكن استخدامه داخلها . ويطبق هذا المفهوم على كل أنواع التوضيحات ، وليس على الثوابت والمتغيرات وحدها .

### مثال (٧-٤)

أكبر عدد من ثلاثة أعداد . دعنا نفحص مرة أخرى برنامج البسكال الموجود فى مثال (٧ - ٣)

```
PROGRAM sample(input,output);
(* This program uses a procedure to determine the maximum
   in each set of three integer quantities *)
VAR a,b,c : integer;

PROCEDURE maximum;
(* This procedure finds the maximum
   of three integer quantities *)
VAR max : integer;
BEGIN
  IF a > b THEN max := a ELSE max := b;
  IF c > max THEN max := c;
  writeln(' The maximum is ',max)
END;

BEGIN (* main action block *)
  readln(a,b,c);
  WHILE a <> 0 DO
    BEGIN maximum; readln(a,b,c) END
  END.
```

يحتوى هذا البرنامج على إجراء واحد يسمى maximum . وتوضح المتغيرات a , b , c خارج هذا الإجراء . وعلى هذا ... فهي شاملة بالنسبة لهذا الإجراء . ويمكن استخدام هذه المتغيرات داخل الإجراء وخارجه ( كما هو الحال فى هذا المثال ) . وعلى أية حال ... فإن المتغير max تم توضيحه داخل الإجراء . وعلى هذا ... فإن max متغير محلي لهذا الإجراء ، ولا يمكن استخدامه خارجه .

وبصفة عامة ... تفضل المعرفة المحلية عن المعرفة الشاملة . وهذا لا يتعارض مع المنطق الشامل للبرنامج . ويسهم استخدام المعرفة المحلية بصفة خاصة فى جعل البرنامج أكثر وضوحا ، كما أنه يقلل أيضا من فرصة حدوث أخطاء برمجية ، بسبب أدلة الإشارة غير الصحيحة أو غير المتناسقة .

ومن ناحية أخرى ... يتطلب الكثير من البرامج أن تميز عناصر بيانات محددة داخل وخارج الإجراء . وإحدى طرق نقل مثل هذه المعلومات عبر حدود البرنامج هي استخدام المعارف الشاملة ، حيث يمكن الإشارة إلى المعارف الشاملة من أى مكان كلما دعت الحاجة لذلك . ( وهناك طريقة أخرى ، وهي استخدام مؤشرات parameters كما يتضح ذلك من القسم التالى ) . وعلى هذا ... يمكن أن يحتوى البرنامج على كل من المعارف الشاملة والمحلية ، طبقا للمتطلبات المنطقية للمشكلة وللتكوين المناظر للبرنامج . ومن المهم تمييز وقت استخدام كل نوع من أنواع المعارف

ومن الممكن استخدام نفس المعارف لتمثيل كيانات مختلفة فى أجزاء مختلفة من أجزاء البرنامج ( بالرغم من أن هذا يعتبر برمجة ضعيفة عمليا ) . وعلى هذا ... فالمعرف الذى يتم توضيحه بأنه محلى داخل الإجراء ، يمكن أن يكون له نفس الاسم كمعرف شامل يوضح خارجيا . وفى هذه الحالات فإن التعريف المحلى يكون له الأولوية داخل مدى المعارف . وخارج هذا المدى لا يكون المعارف المحلى معروفا ، ولا يطبق إلا التعريف الشامل .

### مثال (٧-٥)

والشكل البيكل التالى لبرنامج يسكال يحتوى على إجرائين :

```
PROGRAM sample(input,output);
VAR a,b : integer;
    c,d : char;

    PROCEDURE one;
    VAR a,d : real;
    BEGIN
        .
        .
        .
    END;

    PROCEDURE two;
    VAR a : char;
        b : boolean;
    BEGIN
        .
        .
        .
    END;

BEGIN (* main program - action statements *)
    .
    .
    .
END.
```

لاحظ أن الإجراء الأول يحتوى على متغيرين حقيقيين محليين ، هما : a , d . ويمكن لهذا الإجراء أن يشير أيضا إلى متغيرين شاملين ، هما : b وهو صحيح ، و c وهو حرفى .

ويميز الإجراء الثانى المتغيرين المحليين a , b ، وهما من النوع الحرفى والبوليان على التوالى . كما يمكن للإجراء الثانى أن يشير أيضا إلى المتغيرين c , d وهما حرفيان .

وتحتوى المجموعة الأساسية للبرنامج على أربعة متغيرات محلية a , b ، وهما صحيحان ، و c , d وهما حرفيان .

لاحظ أن العديد من أسماء المتغيرات يستخدم بطريقة مختلفة داخل الإجراءات ، عنها في الجزء الرئيسى للبرنامج . وبصفة خاصة تم إعادة تعريف a داخل كل إجراء ، وتم إعادة تعريف كل من b , d داخل إجراء واحد من الإجرائين . وتأخذ التعريفات المحلية الأولوية داخل الإجراءات المناظرة لها . أما داخل الجزء الرئيسى من البرنامج ، فسوف تفسر المتغيرات طبقاً لتوضيحها الأصلي .

ويطبق موضوع التوضيحات المحلية والشاملة على الإجراءات ( والدوال ) وعلى الثوابت والمتغيرات . وعلى هذا ... فمن الممكن توضيح إجراء ( أو دالة ) داخل إجراء آخر ، وتسمح هذه السمة بتداخل الإجراءات داخل بعضها . وفى مثل هذه الحالات يكون من المهم بالطبع ألا يحدث اتصال بالإجراء خارج المجموعة التى تحتوى على توضيح الإجراء نفسه .

### مثال (٦-٧)

فيما يلى شكلاً هيكلياً لبرنامج بسكال يحتوى على إجراءات متداخلة . ومدى كل إجراء مختلف موضح عن طريق الترحيل .

```
PROGRAM main(input,output);

  PROCEDURE one;

    PROCEDURE two;
    BEGIN (* procedure two - action statements *)
      .
      .
      .
    END;

    PROCEDURE three;
    BEGIN (* procedure three - action statements *)
      .
      .
      two; (* reference to procedure two *)
      .
      .
    END;

    BEGIN (* procedure one - action statements *)
      .
      .
      two; (* reference to procedure two *)
      three; (* reference to procedure three *)
      .
      .
    END;

  BEGIN (* main program - action statements *)
    .
    .
    one; (* reference to procedure one *)
    .
    .
  END.
```

فى هذا المثال نجد أن الإجراء الأول موضح داخل الجزء الرئيسى للبرنامج . أما الإجراءان three , two , one فموضحان داخل الإجراء one . وعلى هذا فإن مدى الإجرائين three , two يكون محليا للإجراء one . وعلى هذا ... فىمكن الاتصال بالإجرائين three , two داخل الإجراء one ، وليس داخل العبارات الاجرائية للجزء الرئيسى للبرنامج . كما يمكن للإجرائين three , two أن يتصلا ببعضهما أيضا . ومن ناحية أخرى ... يمكن الاتصال بالإجراء one بواسطة العبارات الإجرائية للجزء الرئيسى للبرنامج ، حيث إنها موضحة داخل هذا الجزء .

### مثال ( ٧ - ٧ )

أكبر عدد من ثلاثة أعداد . نعتبر الآن صيغة مختلفة لبرنامج البسكال الموجود فى مثال ( ٧ - ٤ ) . وفى هذا المثال نجزئ الإجراء الاصلى maximum إلى إجرائين ، أحدهما داخل الآخر ( كما هو موضح بواسطة الترحيل ) . ويسمى الإجراء الخارجى بنفس الاسم maximum ، إلا أن بعض الجمل الإجرائية ظهرت الآن داخل الإجراء الداخلى findmax .

```
PROGRAM sample(input,output);

(* This program uses a procedure to determine the maximum
   in each set of three integer quantities *)

VAR a,b,c : integer;

PROCEDURE maximum;
(* This procedure finds the largest
   of three integer quantities *)
VAR max : integer;

PROCEDURE findmax;
(* This is where the action is *)
BEGIN
  IF a > b THEN max := a ELSE max := b;
  IF c > max THEN max := c
END;

BEGIN (* back to maximum - action statements *)
  findmax;
  writeln(' The maximum is ',max)
END;

BEGIN (* main action statements *)
  readln(a,b,c);
  WHILE a <> 0 DO
    BEGIN maximum; readln(a,b,c) END
  END.
```

لاحظ أن findmax تم توضيحه داخل maximum ، كما أن maximum تم توضيحه داخل الجزء الرئيسى للبرنامج . وعلى هذا ... فإن findmax يكون محليا لإجراء maximum . لاحظ أيضا أنه يتم الاتصال بإجراء maximum من الجزء الرئيسى للبرنامج ، بينما يتم الاتصال بإجراء findmax من داخل الإجراء maximum .

وتظهر حالة شبيهة بالنسبة للمتغيرات a - b - c ، max ، c - b - a . فالمتغيرات a - b - c بصفة خاصة هى شاملة بالنسبة للإجرائين ، حيث إنه تم توضيحها داخل الجزء الرئيسى للبرنامج . ويمكن استخدام هذه المتغيرات الثلاثة على أية حال فى أى مكان فى البرنامج . ومن ناحية أخرى ... فإن المتغير max يعد متغيراً محلياً للإجراء maximum ويمكن على ذلك استخدامه داخل الإجراء maximum ، أو داخل الإجراء الداخلى findmax فقط .

وأخيرا يجب ملاحظة مفهوم المدى مع عناوين العبارات ومع الثوابت والمتغيرات والإجراءات (والدوال) . ولا يمكن وضع عنوان بصفة خاصة لعبارة إلا إذا وجدت العبارة داخل مدى توضيح العنوان . كما أنه لا يمكن نقل التحكم إلى عبارة لها عنوان من خارج مدى هذا العنوان . وأنه من الممكن على أية حال نقل التحكم إلى عبارة لها عنوان من أى مكان آخر داخل مدى العنوان . وعلى هذا ... يمكن نقل التحكم خارج الإجراء (أو الدالة) إذا ما حدث النقل خارج مدى تعريف العنوان . وهذا يطبق أيضا على الإجراءات المتداخلة لأى عدد من مرات التداخل .

## مثال (٧-٨)

حساب متوسط قائمة من الأعداد . فى مثال ٦-٢٦ رأينا برنامج يسكال يحسب متوسط قائمة بالأعداد ، وذلك باستخدام عبارة GOTO لنقل التحكم خارج دورة الشرط عندما يقرأ الكمبيوتر عددا سالباً . (العدد السالب يعبر عن تحقق شرط التوقف ، ولا يحسب فى حساب المتوسط) .

ولنعد كتابة هذا البرنامج الآن ، بحيث إن بيانات المدخلات ودوال حساب البيانات تعامل عن طريق إجراء . وسوف نظل محتفظين باستخدام عبارة GOTO ، بحيث ينقل التحكم خارج الإجراء عند إدخال عدد سالب داخل الكمبيوتر . وفيما يلي برنامجا كاملا بلغة البسكال لأداء ذلك .

```
PROGRAM average6(input,output);

(* THIS PROGRAM CALCULATES THE AVERAGE OF A LIST OF NUMBERS
   USING THE WHILE - DO STRUCTURE AND A GOTO STATEMENT
   WITHIN A PROCEDURE *)

LABEL 10;
VAR count : integer;
    sum,average : real;

PROCEDURE enterdata;
(* This procedure reads and sums
   successive real, positive quantities *)
VAR x : real;
    flag : boolean;
BEGIN (* action statements *)
    flag := true;
    WHILE flag DO
        BEGIN
            read(x);
            IF x < 0 THEN GOTO 10;
            sum := sum+x;
            count := count+1
        END
    END;

BEGIN (* main action statements *)
    count := 0;
    sum := 0;
    enterdata;
10 : average := sum/count;
    writeln(' The average of ',count:5,' numbers is ',average)
END.
```

لاحظ أن العبارة رقم 10 والمتغيرات count , sum , average تم توضيحها داخل الجزء الرئيسى للبرنامج ، حيث إن هذه العناصر مطلوبة فى كل من جزء الأداء لهذا الجزء ، وفى الإجراء أيضا ، ومن ناحية أخرى ... فإن flag و x تم توضيحهما داخل الإجراء محليا ، وذلك نظرا لأن هذه المعرفات غير مطلوبة خارج الإجراء . ولاحظ أن التحكم ينقل خارج الإجراء مباشرة عند ظهور قيمة سالبة للمعرف x .

ويجب أن يكون مفهوما أن هذا المثال يحتوى على العديد من الأشياء المريحة لتوضيح سمات معينة . وفى الواقع العملى يجب أن يكتب هذا البرنامج دون استخدام عبارة GOTO ، والأكثر من هذا ... يجب أن تظهر حسابات المتوسط وعبارات المخرجات التالية لها داخل الإجراء . وهذا يسمح بتوضيح كل المتغيرات داخل الإجراء محليا . وهو اتجاه أفضل لحل المشكلة .

### 3. PARAMETERS

### ٣ - المؤشرات :

يتطلب الكثير من برامج البسكال أن يحدث تبادل للمعلومات بين إجراء ( أو دالة ) وبين النقطة التى حدث عندها إشارة للإجراء ( أو للدالة ) ، وإحدى طرق تحقيق ذلك هى استخدام المتغيرات الشاملة كما سبق ذكره فى القسم السابق . كما سبق أن رأينا على أية حال أنه هناك وجهات نظر لاتحيب من استخدام المتغيرات الشاملة . فمثلا يمكن لتغيير قيمة متغير شامل داخل أحد الإجراءات أن ينتج عنها تغيير معلومات أخرى خارج الإجراء سهوا ، والعكس صحيح أيضا . وأكثر من هذا ... فإن نقل مجموعات بيانات متعددة لايمكن تحقيقه بسهولة باستخدام المتغيرات الشاملة .

ويقدم استخدام المؤشرات طريقة أفضل لتبادل المعلومات بين إجراء والنقطة المشيرة إليه ، فينقل كل عنصر بيانات بين المؤشر الفعلى actual parmeter والموجود داخل دليل ( أو المشير إلى ) الإجراء والمؤشر الرسمى formal parmeter المناظر والمعرف داخل الإجراء نفسه . وعند الاتصال بالإجراء تحل المؤشرات الفعلية محل المؤشرات الرسمية منتجة آلية تبادل المعلومات بين الإجراء ونقطة الإشارة إليه . وتعتمد هذه الطريقة التى يحدث بها النقل على طريقة تعريف واستخدام المؤشرات .

### مثال (٧-٩)

فيما يلى تخطيطا هيكليا يوضح أبسط أنواع تبادل المعلومات بين الإجراء وبين دليله :

```
PROGRAM sample(input,output);
VAR a,b,c,d : real;

PROCEDURE flash(x,y : real);
BEGIN
.
.
.
(* process the values of x and y *)
.
.
END;

BEGIN (* main program action statements *)
.
.
flash(a,b);
.
.
flash(c,d);
.
.
END.
```

فى هذا المثال المتغيرات الحقيقية  $x, y$  هى مؤشرات رسمية معرفة داخل الإجراء `flash` . أما المؤشرات الفعلية ، فهى المتغيرات الحقيقية  $a, b, c, d$  . نفترض أنه تم تحديد قيم فى مكان ما فى البرنامج لكل من  $b, c, d$  ، وذلك قبل الإشارة إلى الإجراء .

تتسبب أول إشارة إلى الإجراء فى نقل قيم المؤشرات الفعلية  $a, b$  إلى المتغيرات الرسمية  $x, y$  . وعلى هذا ... فإن قيم  $a, b$  تمر عبر الإجراء `flash` ، حيث يتم تشغيلها هناك . ( وطريقة تشغيلها غير موضحة فى هذا المثال ) .

وتعتمد هذه العملية فى عبارة الإجراء الثانية ، مع نقل قيم  $c, d$  فى هذه المرة إلى  $x, y$  . وعلى هذا ... فتنتقل قيم  $c, d$  عبر الإجراء `flash` ، حيث يتم تشغيلها أيضا .

لاحظ أننا قمنا بتشغيل مجموعتين مختلفتين من البيانات ببساطة ، وذلك عن طريق الاتصال بنفس الإجراء مرتين ، وذلك بمجموعتين مختلفتين من المؤشرات الفعلية فى كل مرة .

وهناك قواعد معينة يجب أن تراعى عند إعداد مناظرة بين دليل ( أو الإشارة إلى ) الإجراء ، وبين الإجراء نفسه . ( أى عند التعويض بالمؤشرات الفعلية فى المؤشرات الرسمية ) ، وهى مايلى :

(١) يجب أن يكون عدد المؤشرات الفعلية فى دليل الإجراء ، هو نفس عدد المؤشرات الرسمية الموجودة فى تعريف الإجراء .

(٢) يجب أن يكون لكل مؤشر فعلى نفس النوع ، مثل المؤشر الرسمى المناظر له .

(٣) يجب التعبير عن كل مؤشر فعلى بطريقة متناسقة مع المؤشر الرسمى المناظر له كما تحدده فئة المؤشر الرسمى ( وفئة المؤشر الرسمى موصوفة أدناه ) .

### مثال (٧-١٠)

اعتبر التكوين الهيكلى للبرنامج التالى :

```
PROGRAM sample(input,output);
VAR a,b : integer;
    c,d : real;

    PROCEDURE flash(x : integer; y : real);
    BEGIN
        .
        .
        (* process the values of x and y *)
        .
        .
    END;

BEGIN (* main program action statements *)
    .
    .
    flash(a,c);
    .
    .
    flash(b,d);
    .
    .
END.
```



لاحظ أن كل دليل إجراء يحتوى على مؤشرين فعليين ، حيث تم تعريف مؤشرين رسميين (  $y, x$  ) داخل الإجراء . وأكثر من هذا ... فإن  $x$  موضح بأنه متغير صحيح ، و  $y$  موضح بأنه متغير حقيقي . وعلى هذا ... فكل دليل لإجراء يجب أن يحتوى على متغير صحيح ومتغير حقيقي ، وبنفس هذا الترتيب .

ويمكن أن يحتوى الإجراء على أربع فئات من المؤشرات الرسمية ، وهى مؤشرات قيمة value parameters ومؤشرات متغير variable parameters ومؤشرات إجراء procedure parameters ومؤشرات دالة function parameters . وسوف نناقش مؤشرات القيمة ومؤشرات المتغير الآن . كما أن مؤشرات الإجراء ومؤشرات الدالة سوف نتناقش فيما بعد فى هذا الفصل ( انظر القسم ٧ - ٥ ) .

## Value Parameters مؤشرات القيمة :

يمكن التفكير فى مؤشرات القيمة بصورة أفضل على أنها مؤشرات مدخلات input parameters ، وذلك بالنسبة إلى إجراءاتها المناظرة لها . ويشمل استخدام مؤشرات القيمة نقل قيمة بدلا من التعويض بالمؤشر الفعلى . وعلى هذا ... فعند نقل المعلومات بين مؤشر فعلى ومؤشر قيمة ، فإن قيمة المؤشر الفعلى تتحدد لمؤشر القيمة . وعند ذلك يمكن تشغيل هذه القيمة داخل الإجراء ( بالإشارة إلى مؤشر القيمة ) والقيم الممثلة بواسطة مؤشرات قيمة لا يمكن على أية حال نقلها فى الاتجاه العكسى ، أى من الإجراء إلى الجزء المنادى على الإجراء من البرنامج . وهذا هو سبب الإشارة إلى مؤشرات القيمة بأنها مؤشرات مدخلات ..

ومؤشرات القيمة سهلة جدا فى استخدامها ... فبتم توضيحها بذكر أسمائها ، وأنواع البيانات المناظرة لها ببساطة داخل عنوان الإجراء ، وذلك دون ذكر أى سابقة prefix ( مثل VAR ) . وغياب مثل هذه السابقة هو الذى يعرف هذه الفئة من المؤشرات ببساطة . وتستخدم مؤشرات القيمة فى كل من مثال ٧ - ٩ و ٧ - ١٠ .

ويجب أن يكون مفهوما أن أى تغيير فى قيمة مؤشر القيمة داخل الإجراء لن يؤثر على قيمة أى مؤشر من المؤشرات الفعلية . ( تذكر أن المؤشرات الفعلية يشار إليها بأنها مؤشرات مدخلات input ) . وهذه الخاصية قد تحد من استخدام مؤشرات القيمة . وعلى أية حال ... فمثل هذه المؤشرات سهلة فى استخدامها فى الحالات التى يسمح فيها بتبادل المعلومات فى اتجاه واحد . وأكثر من هذا ... فحيث إن قيم values المؤشرات الفعلية هى التى تنتقل بدلا من المؤشرات نفسها ، فهناك عمق ممكن فى طريقة كتابة المؤشرات الفعلية . ويمكن التعبير عن المؤشرات الفعلية بصفة خاصة ككوابيت أو متغيرات أو تعبيرات ( مع الحفاظ على أن قيمة المؤشر تكون من النوع المناسب ) .

## مثال (٧-١١)

اعتبر التعديل التالى فى البرنامج الذى سبق توضيح تخطيطه فى مثال ٧ - ١٠

```
PROGRAM sample(input,output);
VAR a,b : integer;
    c,d : real;

PROCEDURE flash(x : integer; y : real);
BEGIN
.
.
.
(* process the values of x and y *)
.
.
END;
```

```
BEGIN (* main program action statements *)
.
.
flash(3,a*(c+d)/b);
.
.
flash(2*(a+b),-0.5);
.
.
END.
```

لاحظ أن المؤشرات الرسمية  $x, y$  تم توضيحها في الإجراء flash ، وأنها مؤشرات قيمة . ( وهذا صحيح أيضا في مثال ٧-٩ و ٧-١٠ ) . وأولها ( $x$ ) هو من النوع الصحيح . أما ثانيهما ( $y$ ) فهو من النوع الحقيقي . وعلى هذا ... فكل إشارة إلى flash يجب أن تحتوى على مؤشرين فعليين ، أولهما يجب أن يكون صحيحا ، وثانيهما يجب أن يكون حقيقيا

ويحتوى الجزء الرئيسى للبرنامج على إشارتين مختلفتين للإجراء ( أى إشارتين مختلفتين لـ flash ) . وتحتوى كل إشارة للإجراء على مؤشرين فعليين ، أولهما من النوع الصحيح ، وثانيهما من النوع الحقيقي كما هو مطلوب . لاحظ أن اثنين من المؤشرات كتبنا كتابتين ، واثنين كتبنا كتعبيرين . وعلى هذا ... ينقل أول دليل ( أو إشارة إلى ) للإجراء القيمة 3 إلى  $x$  والقيمة الخاصة بالتعبير الحقيقي  $a*(c+d)/b$  إلى  $y$  . وبالمثل ينقل الدليل الثانى للإجراء قيمة التعبير الصحيح  $2*(a+b)$  إلى  $x$  وقيمة  $-0.5$  إلى  $y$  .

مثال (٧-١٢)

حساب الاستهلاك . مرة أخرى اعتبر المشكلة الموصوفة في مثال ٦-١٧ الحساب الاستهلاك بإحدى الطرق الثلاث لحساب الاستهلاك . دعنا نعيد الآن كتابة البرنامج مع استخدام إجراء منفصل لكل طريقة من هذه الطرق . يقدم هذا الأسلوب طريقة أوضح لتنظيم البرنامج طبقا لأجزائه المنطقية . ( وفى المثال ٦-١٧ يوجد وصف تفصيلي لمنطق البرنامج ) . وفيما يلي برنامج البسكال كاملا لأداء هذه العمليات .

```
PROGRAM depreciation2(input,output);

(* THIS PROGRAM USES PROCEDURES TO CALCULATE
DEPRECIATION BY ONE OF THREE POSSIBLE METHODS *)

VAR years,choice : integer;
    value : real;
    flag : boolean;

PROCEDURE straightline(n : integer; val : real);
(* Calculate depreciation using the straight line method *)
VAR year : integer;
    deprec : real;
BEGIN
    writeln(' Straight-Line Method');
    writeln;
    deprec := val/n;
    FOR year := 1 TO n DO
        BEGIN
            val := val-deprec;
            write(' End of Year ',year:2);
            write(' Depreciation: ',deprec:5:0);
            writeln(' Current Value: ',val:6:0)
        END
    END
END;
```

(تكملة البرنامج فى الصفحة التالية)

```

PROCEDURE decliningbalance(n : integer; val : real);
(* Calculate depreciation using the
   double-declining balance method *)
VAR year : integer;
    deprec : real;
BEGIN
    writeln(' Double Declining Balance Method');
    writeln;
    FOR year := 1 TO n DO
        BEGIN
            deprec := 2*val/n;
            val := val-deprec;
            write(' End of Year ',year:2);
            write('   Depreciation: ',deprec:5:0);
            writeln('   Current Value: ',val:6:0)
        END
    END;

PROCEDURE sumofyears(n : integer; val : real);
(* Calculate depreciation using the
   sum-of-the-years'-digits method *)
VAR year : integer;
    deprec,tag : real;
BEGIN
    writeln(' Sum-of-the-Years''-Digits Method');
    writeln;
    tag := val;
    FOR year := 1 TO n DO
        BEGIN
            deprec := (n-year+1)*tag/(n*(n+1)/2);
            val := val-deprec;
            write(' End of Year ',year:2);
            write('   Depreciation: ',deprec:5:0);
            writeln('   Current Value: ',val:6:0)
        END
    END;

BEGIN (* main action block *)
    flag := true;
    REPEAT
        page;      (* begin input routine *)
        write(' Method: (1-SL 2-DDB 3-SYD 4-End) ');
        readln(choice);
        IF choice <> 4 THEN
            BEGIN
                write(' Original value: ');
                readln(value);
                write(' Number of years: ');
                readln(years);
                writeln
            END;
            (* input routine *)
            CASE choice OF
                1 : straightline(years,value);
                2 : decliningbalance(years,value);

```

(تكملة البرنامج في الصفحة التالية)

```

3 : sumofyears(years,value);
4 : BEGIN
    writeln(' That''s all, folks!');
    flag := false
END
END (* choice *)
UNTIL flag = false
END.

```

لاحظ أن مكون CASE مازال مستخدماً كما في مثال ٦ - ١٧ إلا أن البرنامج يستخدم الآن إجراء مختلفاً لكل نوع من أنواع الحسابات . ويجب أن تستخدم الإجراءات مؤشرات القيمة val , n التي تمثل العمر الافتراضي للعنصر المستهلك وقيمتها الأصلية على التوالي . والمؤشرات الفعلية المناظرة في الجزء الرئيسي للبرنامج هي years , value . وعلى هذا ... فنعد الإشارة إلى الإجراء لتحديد قيمة year للمؤشر n ، وتحديد قيمة value للمؤشر val . (ويمكننا بالطبع استخدام نفس أسماء المتغيرات في كل من الجزء الرئيسي للبرنامج وكل إجراء من الإجراءات . وهذا يكون أبسط ، بالرغم من أنه لا يكون معطياً للمعلومات بطريقة أفضل ) .

لاحظ أن استخدام مؤشرات القيمة مناسب في هذا المثال ، نظراً لأن الإجراءات لا تغير قيم المؤشرات ، ولا تعود بأى قيمة إلى الجزء الرئيسي ( المندى ) للبرنامج .

## Variable Parameters

## مؤشرات المتغير :

لقد رأينا أن مؤشرات القيمة مريحة في العمل في مواقف تحتاج لنقل معلومات من دليل الإجراء إلى الإجراء فقط . وفي العديد من التطبيقات - على أية حال - يجب نقل المعلومات في كلا الاتجاهين بين الإجراء ودليله . وفي كلمات أخرى ... يجب أن يكون الإجراء قادراً على الحصول على مدخلات من الجزء المندى عليه ، وكذلك إعادة مخرجات إلى هذا الجزء . وتستخدم مؤشرات المتغير في هذه الحالات بصفة خاصة .

وعندما يتم الاتصال بإجراء يحتوى على مؤشرات متغير ، يتم التعويض بالمؤشر الفعلى الموجود في الإشارة للإجراء في المتغير الرسمي الموجود داخل الإجراء نفسه . وعلى هذا ... يستخدم المؤشر الفعلى أثناء تنفيذ الإجراء . وهذا بعكس استخدام مؤشر القيمة ، حيث إن قيمة value المؤشر الفعلى هي التي تحدد للمؤشر الرسمي ( لاحظ الفرق بين التحديد assignment والتعويض substitution ) . وعملية التعويض هذه هي التي تسمح بنقل المعلومات في الاتجاهين بين دليل الإجراء والإجراء نفسه .

ومن ناحية أخرى ... تجب الإشارة إلى أنه يمكن تعويض متغير بمتغير آخر فقط . وعلى هذا ... فالمؤشرات الفعلية التي تعوض كمؤشرات متغير يجب أن تكون متغيرات أيضاً ، ولا يمكن أن تكون ثوابت أو تعبيرات . وعلى هذا ... تكون لدينا درجة عمومية أقل في استخدام مؤشرات المتغير عن استخدام مؤشرات القيمة .

وهناك نتيجة أخرى لعملية التعويض ، وهي أن أى تغيير يحدث في قيمة مؤشر المتغير داخل الإجراء سوف يغير من قيمة المؤشر الفعلى المناظر الموجود خارج الإجراء . وعلى هذا ... فيمكن أن تؤثر مؤشرات المتغير على البرنامج ككل ، بالرغم من أن مداها يكون محلياً داخل الإجراء التي تم توضيحها فيه .

وتوضح مؤشرات المتغير داخل عنوان الإجراء كما يحدث بالنسبة لمؤشرات القيمة . وعلى أية حال ... فيجب أن يسبق توضيحات مؤشرات المتغير الكلمة الأساسية VAR كما يتضح ذلك من المثال التالي .

### مثال (٧-١٣)

اعتبر مرة أخرى التكوين الهيكلي للبرنامج الموجود في مثال ٧ - ١٠ إذا ما عدل الإجراء ليستخدم مؤشرات متغير ، فسوف يظهر التكوين الهيكلي على النحو التالي :

```
PROGRAM sample(input,output);
VAR a,b : integer;
    c,d : real;

    PROCEDURE flash( VAR x : integer; VAR y : real);
    BEGIN
        .
        .
        (* process the values of x and y *)
        .
        .
    END;

BEGIN (* main program action statements *)
    .
    .
    flash(a,c);
    .
    .
    flash(b,d);
    .
    .
END.
```

تتسبب أول إشارة للإجراء في تعريف المؤشرات الفعلية a , c مكان x , y . فإذا ما تغيرت قيمة x أو y داخل الإجراء ، فإن التغيير المناظر في قيمة a , c يحدث في الجزء الرئيسي من البرنامج .

وبالمثل ينتج عن ثانی إشارة للإجراء تعريف d , b مكان x , y . وأي تغيير في قيمة x أو y داخل الإجراء يتسبب في تغيير قيمة b أو d في الجزء الرئيسي للبرنامج .

لاحظ أن المؤشرات الفعلية تتفق مع المؤشرات الرسمية المناظرة في عددها وفي نوعها .

### مثال (٧-١٤)

البحث عن أكبر قيمة . افترض أننا نرغب في إيجاد قيمة خاصة من قيم x ، والتي تتسبب في تعظيم الدالة .

$$y = x \cos (x)$$

داخل منطقة محاطة بقيمة  $x = 0$  من ناحية اليسار ، وقيمة  $x = \pi$  من ناحية اليمين . ونطلب أن تكون قيمة x التي تعظم الدالة معروفة بدقة . كما نطلب أيضا أن يكون مخطط البحث كقوسية ، بمعنى أنه يجب أن يتم تقويم الدالة  $y = x \cos (x)$  أقل عدد ممكن من المرات .

وكطريقة واضحة لحل هذه المشكلة ، يجب إنتاج عدد كبير من دوال متقاربة مع بعضها في محاولات البحث ( أي تقويم الدالة عند نقاط  $x = 0.0001$  ,  $x = 0.0002$  , ... ,  $x = 3.1415$  ,  $x = 3.1416$  ) وتحديد أكبر هذه

القيم بالفحص البصرى . وهذه الطريقة منخفضة الكفاءة على أية حال ، وتتطلب تدخلا آدميا للحصول على النتيجة النهائية . بدلا من ذلك ... دعنا نستخدم مخطط الحذف elimination scheme التالى ، وهو إجراء حسابات مرتفع الكفاءة لكل الدوال التى لها نهاية عظمى واحدة داخل منطقة البحث .

وسوف تجرى الحسابات كما يلى . تبدأ بنقطتى بحث عند مركز خطوة البحث ، يقعان على مسافة قريبة جدا من بعضهما كما هو موضوع فى شكل ٧ - ١ . وقد استخدمت الاصطلاحات التالية :

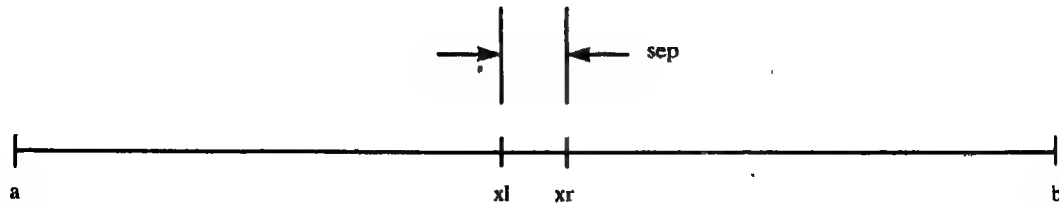
a = النهاية اليسرى لمنطقة البحث .

x<sub>l</sub> = النقطة الداخلية اليسرى للبحث .

x<sub>r</sub> = النقطة الداخلية اليمنى للبحث .

b = النهاية اليمنى لمنطقة البحث .

sep = المسافة بين x<sub>l</sub> , x<sub>r</sub> .



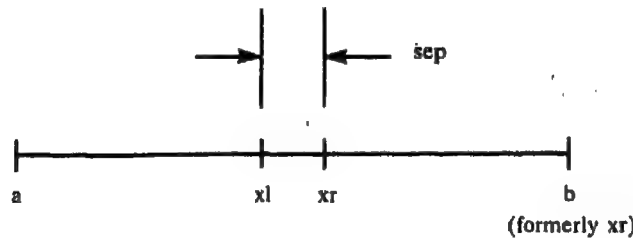
شكل ٧ - ١

فإذا ما عرفت a , b , sep ، فيمكن حساب النقاط الداخلية على النحو التالى :

$$x_l = a + .5(b-a-sep)$$

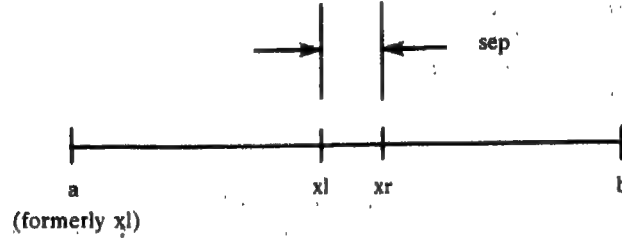
$$x_r = a + .5(b-a+sep) = x_l + sep$$

دعنا نقوم الدالة  $y = x \cos(x)$  عند  $x_l$  ,  $x_r$  ، ونسمى هذه القيم  $y_l$  ,  $y_r$  على التوالى . افترض أن  $y_l$  كانت أكبر من  $y_r$  . هذا يعنى أن أكبر قيمة تقع بين a ,  $x_r$  . وعلى هذا ... فإننا نحتفظ بجزء المنطقة الذى يتراوح من  $x = a$  إلى  $x = x_r$  ( وسوف نشير إلى النقطة  $x_r$  القديمة حاليا بأنها b ، حيث إنها هى نقطة النهاية اليمنى لمنطقة البحث ) ونتتج نقطتين جديدتين للبحث ، وهما :  $x_l$  ,  $x_r$  . وتقع هاتان النقطتان فى مركز منطقة البحث الجديدة بعيدتين عن بعضهما مسافة sep كما هو موضح فى شكل ٧ - ٢ .



شكل ٧ - ٢

ومن ناحية أخرى ... افترض أننا في منطقة بحثنا الأصلي original كانت  $y_r$  أكبر من  $y_l$  . سوف يحدد ذلك أن منطقة البحث الجديدة يجب أن تقع بين  $x_l$  ,  $b$  . وعلى هذا ... فإننا نعيد تسمية النقطة التي كانت تسمى  $x_l$  في الأصل بأنها النقطة  $x$  ، ثم ننتج نقطتي بحث  $x_l$  ,  $x_r$  عند مركز منطقة البحث الجديدة ، كما هو موضح في شكل ٧ - ٣ .



شكل ٧ - ٣

نستمر في إنتاج زوج من نقاط البحث في مركز كل منطقة جديدة ومقارنة قيم  $y$  المناظرة ، وحذف جزء من منطقة البحث ، حتى تصبح منطقة البحث أقل من  $3*sep$  . وعند حدوث ذلك ، فلن نستطيع أن نميز النقاط الداخلية من حدود منطقة البحث . وعند ذلك ينتهي البحث .

وفي كل مرة تحدث مقارنة بين  $y_l$  ,  $y_r$  نحذف منطقة البحث التي تحتوى على أصغر قيمة من قيم  $y$  . فإذا ما حدث أن كانت قيمتا  $y_l$  ,  $y_r$  متساويتين ( الشئ الذى يمكن أن يحدث ، إلا أن حدوثه غير عادى ) تتوقف عملية البحث ، حيث يفترض حدوث أكبر قيمة للدالة عند هذه النقطة التي تقع بين  $y_l$  ,  $y_r$  .

وعند الانتهاء من البحث سواء أكان ذلك لصغر منطقة البحث صغرا كافيا أم لتساوى قيمتي  $y_l$  ,  $y_r$  ، يمكننا حساب الموقع التقريبي لأكبر قيمة كمايلي :

$$x_{max} = .5*(x_l + x_r)$$

والقيمة القصوى المناظرة للدالة يمكن الحصول عليها كمايلي :  $x_{max} \cos(x_{max})$

دعنا نعتبر تخطيطا لبرنامج لحالة عامة ، حيث  $a$  ,  $b$  هي كميات مدخلات ، و  $sep$  لها قيمة ثابتة هي 0.0001

(١) تحدد قيمة ثابتة للمتغير  $sep$  بأنها  $sep = 0.0001$  .

(٢) تقرأ قيم  $a$  ,  $b$  داخل الكمبيوتر .

(٣) يتكرر مايلي حتى تصبح  $y_l$  مساوية  $y_r$  ( حيث تقع القيمة القصوى بينهما ) أو تكون أحدث قيمة للفرق بين  $a$  ,  $b$  (  $b-a$  ) أقل من أو تساوى  $3*sep$  .

(١) تنتج نقطتان داخليتان  $x_l$  ,  $x_r$  .

(ب) تحسب قيمتان  $y_l$  ,  $y_r$  مناظرتان لهما ، ويتحدد أيهما أكبر .

(ج) نقلل منطقة البحث بحذف الجزء الذى لا يحتوى على أكبر قيمة من قيم  $y$  .

(٤) تقوم  $x_{max}$  ,  $y_{max}$  .

(٥) تكتب قيم  $x_{max}$  ,  $y_{max}$  ويتوقف العمل .

ويمكن أن تعد الخطوة رقم ٣ على أنها إجراء . ولعمل ذلك ، دعنا نعرف المتغيرات  $a$  ,  $b$  ,  $x_l$  ,  $x_r$  ,  $y_l$  ,  $y_r$  بأنها مؤشرات متغير . ( والقيم التي تمثلها هذه المؤشرات سوف تتغير أثناء الحسابات ويعاد نقلها للأمام وللخلف بين الإجراء والجزء الرئيسى للبرنامج ) .

```

PROCEDURE reduce( VAR a,b,xl,xr,yl,yr : real);
(* Interval reduction routine *)
BEGIN
  xl := a + 0.5*(b-a-sep);
  xr := xl + sep;
  yl := xl*cos(xl);
  yr := xr*cos(xr);
  IF yl > yr THEN b := xr;  (* retain left interval *)
  IF yr > yl THEN a := xl  (* retain right interval *)
END;
```

لاحظ أن sep لم يوضح بأنه مؤشر داخل الإجراء . وسوف نعرف sep كثابت شامل داخل الجزء الرئيسى للبرنامج ، حيث إن قيمته لن تتغير أبداً ، ولاحظ أيضاً استخدام الدالة القياسية cos داخل الإجراء ، كما أننا نستخدم هذه الدالة أيضاً فى الجزء الرئيسى من البرنامج . وأخيراً فإننا نشير إلى أن تقليل المنطقة الخاصة بالبحث لن يعمل بصورة مناسبة إذا ما كانت بدلا  $y_l, y_r$  متساويتين ( هذا ممكن ، إلا أنه بعيد الحدوث ) . ويمكن تصحيح هذه المشكلة ببساطة ، وذلك بوضع  $>$  من إحدى العلاقات  $>$  .

ومن السهل الآن كتابة البرنامج الكامل كما يلى :

```

PROGRAM maximum1(input,output);

(* THIS PROGRAM FINDS THE MAXIMUM OF A
   FUNCTION WITHIN A SPECIFIED INTERVAL *)

CONST sep = 0.0001;
VAR a,b,xl,xr,xmax,yl,yr,ymax : real;

PROCEDURE reduce( VAR a,b,xl,xr,yl,yr : real);
(* Interval reduction routine *)
BEGIN
  xl := a + 0.5*(b-a-sep);
  xr := xl + sep;
  yl := xl*cos(xl);
  yr := xr*cos(xr);
  IF yl > yr THEN b := xr;  (* retain left interval *)
  IF yr > yl THEN a := xl  (* retain right interval *)
END;

BEGIN (* main action block *)
  readln(a,b);
  REPEAT
    reduce(a,b,xl,xr,yl,yr)
  UNTIL (yl=yr) OR ( b-a <= 3*sep );
  xmax := 0.5*(xl+xr);
  ymax := xmax*cos(xmax);
  writeln(' xmax = ',xmax:8:6, '   ymax = ',ymax:8:6)
END.
```

وينتج عن تنفيذ البرنامج بقيم  $a = 0$  ,  $b = 3141593$  المخرجات التالية

xmax = 0.860586    ymax = 0.561096



وعلى هذا ... يكون لدينا موقع القيمة القصوى والقيمة القصوى نفسها داخل المنطقة الأصلية المعطاء .

#### 4. FUNCTIONS

#### ٤ - الدوال :

الدالة function هي برنامج ذاتي تشبه الإجراء في العديد من الأشياء . ( تذكر مناقشتنا الموجهة لهذا الموضوع في قسم ٢ - ١٢ ) . وعلى عكس الإجراء ، تستخدم الدالة - على أية حال - للعودة بقيمة واحدة بسيطة النوع إلى نقطة الإشارة إلى الدالة . وأكثر من هذا ... فيشار إلى الدالة بتحديد اسمها داخل تعبير ، كما لو كانت متغيراً من النوع البسيط . ويمكن أن يلي اسم الدالة مؤشر فعلى واحد أو أكثر محصوراً أو محصورين بين قوسين ، ومنفصلين بواسطة فواصل . وفي معظم الحالات تنقل المؤشرات الفعلية معلومات لتقويم المؤشرات داخل الدالة . ويمكن على ذلك أن تكون ثوابت أو متغيرات أو تعبيرات .

#### مثال (٧-١٥)

افترض أن factorial هو اسم دالة تحسب مضروب بعض الكميات الصحيحة ( تذكر أن مضروب الكمية الصحيحة n معروف بأنه  $n! = 1 \times 2 \times \dots \times n$  ) . وعلى هذا ... يمكن تقويم العلاقة

$$f = x!/a!(x - a)!$$

كما يلي :

```
f := factorial(x)/(factorial(a)*factorial(x-a));
```

حيث  $a, x, f$  هي متغيرات من النوع الصحيح .

لاحظ التشابه بين استخدام هذه الدالة التي يعرفها المستخدم ، واستخدام الدوال القياسية التي سبق ذكرها في قسم ٢ - ١٢ .

وتحتوي الدالة نفسها على عنوان للدالة وتسم للدالة . ويكتب عنوان الدالة على النحو التالي :

FUNCTION name : type

أو إذا ما استخدمت مؤشرات ، فإنه يأخذ الشكل التالي :

FUNCTION name(formal parameters) : type

ويحدد آخر عنصر type نوع البيانات التي تنتج من الدالة ، وتعود إلى نقطة الإشارة إلى الدالة .

وبصفة عامة ... فإن المؤشرات الرسمية تكون مؤشرات قيمة ، بدلا من كونها مؤشرات متغير . وهذا يسمح بالمؤشرات الفعلية المناظرة بأن تكون ثوابت أو متغيرات أو تعبيرات . ( تذكر أن هذه المؤشرات تنتج قيم مدخلات للدالة فقط . أما القيمة الفردية لمخرجات الدالة ، فسوف يمثلها اسم الدالة وليس المؤشر ) .

ومجموعة الدالة تشبه مجموعة الاجراء ، وتحتوى على نفس قواعد المدى مثل الإجراء . وداخل مجموعة الدالة يجب أن تحدد قيمة من نوع مناسب ( كما هو محدد فى عنوان الدالة ) للمعرف الذى يمثل اسم الدالة . وهذه هى القيمة التى تعود بها الدالة إلى نقطة الإشارة إليها . ويمكن تحديد قيم لاسم الدالة عند نقطتين أو أكثر داخل مجموعة الدالة وعندما يتم التحديد فلا يمكن تغييره .

### مثال (٧-١٦)

فيما يلى دالة اسمها factorial تحسب مضروب كمية صحيحة :

```
FUNCTION factorial(n : integer) : integer;
(* Calculate the factorial of n *)
VAR factor,product : integer;
BEGIN
  IF n <= 1 THEN factorial
    ELSE BEGIN
      product := 1;
      FOR factor := 2 TO n DO
        product := product*factor;
      factorial := product
    END
END;
```

أول سطر يحتوى على الكلمة الأساسية FUNCTION هو عنوان الدالة . لاحظ أن العنوان يحتوى على توضيح لمؤشر قيمة . لاحظ أيضا آخر عنصر فى السطر integer ، والذى يحدد أن الدالة سوف تعود بكمية صحيحة النوع .

تقبل هذه الدالة قيمة من قيم n ، ثم تحسب قيمة مضروب n باستخدام متغيرين صحيحين محليين هما product factor ، وتحدد النتيجة النهائية للمعرف factorial وهو اسم الدالة نفسها .

لاحظ أن هناك تحديدين مختلفين لـ factorial ، إلا أن أحدهما فقط هو الذى يستخدم عند تنفيذ الدالة . ويعتمد الاختيار على القيمة التى تحدد لـ n ، وبمجرد تحديد قيمة factorial ، لا تتغير هذه القيمة داخل الدالة .

ولتوضيح هذه النقطة الأخيرة اعتبر الشكل المختلف التالى للدالة السابقة .

```
FUNCTION factorial(n : integer) : integer;
(* Calculate the factorial of n *)
VAR factor : integer;
BEGIN
  factorial := 1;
  IF n > 1 THEN FOR factor := 2 TO n DO
    factorial := factorial*factor
  END;
```

على السطح تبدو هذه الصيغة أكثر وضوحا عن الصيغة الأصلية ، حيث إنها أبسط . وهذه الصيغة ليست صحيحة على أية حال ، أن قيمة factorial تتغير بعد التحديد الأولى عندما تكون n أكبر من 1 .

وتوضح الدوال بنفس الطريقة التى توضح بها الإجراءات داخل الجزء المنادى ، أو فى أى جزء خارجى محيط بالجزء المنادى . وليس هناك ترتيب خاص بالنسبة للدوال والإجراءات . ( تذكر أنه يجب أن تكون توضيحات الدوال والإجراءات هى آخر توضيحات داخل المجموعة ، وتأتى بعد أى توضيحات لعناوين أو ثوابت أو أنواع أو متغيرات ) .

### مثال (٧-١٧)

حساب المضروب ، فيما يلي برنامج بسكال كاملاً ، يحدد مضروب كمية مدخلات معطاه .

```
PROGRAM factorials2(input,output);

(* THIS PROGRAM USES A FUNCTION TO CALCULATE
   THE FACTORIAL OF A GIVEN INTEGER QUANTITY *)

VAR x : integer;

FUNCTION factorial(n : integer) : integer;
(* calculate the factorial of 'n *)
VAR factor,product : integer;
BEGIN
  IF n <= 1 THEN factorial := 1
  ELSE BEGIN
    product := 1;
    FOR factor := 2 TO n DO
      product := product*factor
    factorial := product
  END
END;

BEGIN (* main action block *)
  write(' Enter a positive integer: ');
  readln(x);
  writeln;
  writeln(' x= ',x,' x! = ',factorial(x))
END.
```

لاحظ أنه يتم الاتصال بالدالة factorial مرة واحدة فقط في عبارة writeln داخل الجزء الرئيسي من البرنامج .

عند تنفيذ هذا البرنامج يجب أخذ الاحتياط للقيمة المحددة لـ x ، حيث إنه من الممكن حدوث سريان زائد إذا ما حددت قيمة كبيرة جداً لـ x . ( تذكر أن المضروب يتزايد بسرعة جداً في قيمته ) . وهذا صحيح بصفة خاصة بالنسبة لمعظم أجهزة الكمبيوتر الصغيرة ، حيث تكون أكبر كمية صحيحة مسموح بها صغيرة نسبياً .

يمكن للدوال أن تتداخل واحدة داخل الأخرى بنفس طريقة تداخل الإجراءات . وأكثر من هذا ... يمكن أن يحدث تبادل في تداخل الدوال والإجراءات . وعلى هذا ... يمكن توضيح الدالة داخل إجراء يكون قد سبق توضيحه داخل دالة أخرى ، وهكذا .

### مثال (٧-١٨)

محاكاة مباراة الفرصة Simulation of a Game of Chance . فيما يلي برنامجاً ممتعاً لمشكلة تستخدم إجراء مع دوال متداخلة .

اللعبة هي لعبة نرد ، يتم فيها قذف زوج من النرد مرة واحدة أو أكثر ، حتى يحدث مكسب أو خسارة للشخص الذي يلقى النرد . ويمكن استخدام الكمبيوتر لأداء المباراة عن طريق تعويض إنتاج الأعداد العشوائية للقذف الفعلي للنرد .

وهناك طريقتان للكسب في هذه اللعبة . يمكنك أن تلقى زوج النرد مرة واحدة ؛ وتحصل على 7 أو 11 ، أو يمكنك الحصول على 4 أو 5 أو 6 أو 8 أو 9 أو 10 في أول مرة لإلقاء النرد ، وبعد ذلك فإنك تكرر الحصول على نفس الرقم في إلقاء النرد مرة تالية قبل أن تحصل على الرقم 7 . وهناك طريقتان أيضا للخسارة . يمكنك أن تلقى النرد مرة وتحصل على 2 أو 3 أو 12 ، أو يمكنك أن تحصل على 4 أو 5 أو 6 أو 8 أو 9 أو 10 في أول مرة لإلقاء النرد ، ثم تحصل بعد ذلك على 7 في المحاولة التالية قبل أن تكرر الإلقاء في محاولة للحصول على الرقم الأصلي .

دعنا نستخدم الكمبيوتر في أداء هذه المباراة بطريقة تقليدية ، بحيث يمكن محاكاة الشخص الذي يلقي النرد في كل مرة تضغط على مفتاح return على لوحة مفاتيح الكمبيوتر . عند ذلك تظهر رسالة على الشاشة تحدد ناتج كل عملية لإلقاء النرد . وفي نهاية كل مباراة سوف يسألك الكمبيوتر ما إذا كنت تريد أن تستمر في اللعب أم لا .

يتطلب البرنامج منتج أعداد عشوائية لإنتاج أعداد عشوائية موزعة توزيعاً منتظماً تقع بين الصفر و 1 . ( ونعني بالتوزيع المنتظم uniformly distributed أن أي عدد يقع بين 0 , 1 يمكن أن يظهر بنفس الفرصة التي يظهر بها أي عدد آخر يقع في نفس المنطقة ) . وبعض صيغ البسكال يوجد بها منتج أعداد عشوائية كدالة قياسية أو كإجراء قياسي ، إلا أن هذا لا يتواجد مع كل الصيغ . وعلى هذا ... فسوف نقدم منتج أعداد عشوائية ، مستخدمين أسلوباً معروفاً جيداً ، وهو طريقة power - residue method .

والأعداد التي تنتجها ليست في واقع الأمر عشوائية ، وذلك نظراً لأن نفس تسلسل الأعداد ينتج دائماً لنفس القيمة الابتدائية ( أي لنفس البذرة seed ) . وعلى أية حال ... فسوف يظهر تسلسل الأعداد على أنه عشوائي ، وسوف يكون له العديد من خواص الأعداد العشوائية الفعلية .

وإن نقاش أساس منتج الأعداد العشوائية ، حيث إن ذلك لا يقع في إطار محتويات هذا الكتاب ، إلا أن جزء البرنامج الفعلي بسيط . فنحن نقدم ببساطة منتج الأعداد العشوائية في صيغة دالة تسمى rand . وتكتب هذه الدالة خصيصاً للكمبيوتر الذي يستخدم كميات صحيحة تكتب في 16 بت . ( ومعظم أجهزة الميكروكمبيوتر تقع داخل هذه الفئة ) .

```
FUNCTION rand : real;
(* 16-bit random number generator *)
VAR y,z : integer;
BEGIN
  z := 259*x;
  IF z >= 0 THEN y := z
    ELSE BEGIN y := z + 32767; y := y + 1 END;
  rand := 0.3051757E-4*y;
  x := y
END;
```

وحتى يمكن استخدام هذه الدالة ، يجب أن تحدد قيمة صحيحة لمؤشر المتغير x في كل مرة تستدعي فيها الدالة . وفي البداية تكون هذه هي البذرة seed التي يمكن أن تحدد كمؤشر مدخلات أو كتابت . والإشارات التالية للدالة rand سوف تستخدم القيمة السابقة لـ y كقيمة جديدة لـ x ( لاحظ أن هذه السمة موجودة في نهاية الدالة ) .

دعنا نعرف دالة أخرى تسمى throw لمحاكاة إلقاء النرد مرة واحدة . تحتوي هذه الدالة على منتج أعداد عشوائية . سوف يحاكي كل نرد على حدة بإنتاج رقم صحيح عشوائي يأخذ القيمة 1 , 2 , 3 , 4 , 5 , 6 بنفس درجة احتمال حدوث كل رقم من هذه الأرقام ، ثم تضاف القيمتان الخاصتان بكل نرد معا لتمثلاً لإلقاء زوج النرد مرة واحدة ، ويمكن تمثيل هذه الدالة بالشجرة الشبيهة على النحو التالي : ( انظر الصفحة التالية )

```

FUNCTION throw : integer;
(* Throw the dice one time *)
VAR d1,d2 : integer;

(* define the random number generator *)

BEGIN
    d1 := 1 + trunc(6*rand);
    d2 := 1 + trunc(6*rand);
    throw := d1 + d2
END;

```

وتحتاج هذه الدالة إلى بعض التوضيح . اعتبر على سبيل المثال أول عبارة تحديد ( d1 := ... ) . الدالة rand (x) تعود برقم عشوائي يقع بين الحدين 0 , 1 ( بين 0 و .... 0.999999 بالفعل ) . وعلى هذا ... فإن التعبير trunc (6\*rand) يعود بقيمة عدد عشوائي يقع بين 0 , 6 ( بين 0 و .... 5.999999 بالفعل ) ، والتعبير trunc (6\*rand) (x) يعود برقم صحيح عشوائي ، يمكن أن تكون قيمته 0 , 1 , 2 , 3 , 4 , 5 بنفس درجة احتمال حدوث أى منها . وبعد ذلك نضيف 1 لنحصل على الرقم العشوائي الذى يمكن أن يكون 1 , 2 , 3 , 4 , 5 , 6 بنفس درجة احتمال حدوث أى منها . وعلى هذا ... فإن هذا التعبير يحاكي إلقاء النرد .

دعنا نعرف الآن إجراء يسمى Play ، يمكنه أن يحاكي مباراة كاملة ( أى إلقاء النرد أى عدد لازم للمكسب أو للخسارة ) . وعلى هذا ... يحتوى ذلك الإجراء على دالة throw . ويجب أن يكون داخل هذا الإجراء كل قواعد اللعبة . ويمكننا أن نكتب هذه العمليات بالشفرة الشبيهة على النحو التالى :

```

PROCEDURE play;
(* Simulate a single game of craps *)
VAR score,tag : integer;

(* define function throw *)

BEGIN

    (* instruct the user to throw the dice *)

    score := throw;
    CASE score OF
        7,11 : BEGIN (* win on first throw *)

            (* write a message indicating a win on the first throw *)

            END;
        2,3,12 : BEGIN (* loss on first throw *)

            (* write a message indicating a loss on the first throw *)

            END;
        4,5,6,8,9,10 : BEGIN (* multiple throws *)
            tag := score;
            REPEAT

```

(تكملة البرنامج فى الصفحة التالية)

```

        (* instruct the user to throw the dice again *)

        score := throw;
        UNTIL (score = tag) OR (score = 7);
        IF score = tag
        THEN BEGIN

            (* write a message indicating a win *)

            END
        ELSE BEGIN

            (* write a message indicating a loss *)

            END
        END (* multiple throws *)
    END (* case *)
END;

```

أخيرا يستخدم الجزء الرئيسى من البرنامج للتحكم فى تنفيذ المباراة . وسوف يحتوى هذا الجزء على القليل من المدخلات والمخرجات المتداخلة واستدعاء الإجراء play . ( لاحظ على أية حال أن play سوف يعرف داخل هذا الجزء ) . وعلى هذا ... يمكننا كتابة الشفرة الشبيهة على النحو التالى :

```

PROGRAM craps(input,output);
(* Interactive simulation of a game of craps *)
CONST seed = 12345;
VAR count,x : integer;
    n : real;
    answer : char;
    flag : boolean;

(* define procedure play *)

BEGIN
    x := seed;
    flag := true;

    (* generate a welcoming message *)

    WHILE flag DO
        BEGIN
            play;
            write(' Do you want to play AGAIN? (Y/N) ');
            readln(answer);
            IF (answer = 'N') OR (answer = 'n') THEN flag := false
        END;

        (* generate a sign-off message *)
    END.

```

لاحظ أن البرنامج سوف يتكرر تنفيذه حتى يحدد اللاعب رغبته في إيقافه . وفيما يلي برنامج بسكال كاملاً لهذه المباراة ، وبه بعض التعليقات الإضافية :

```

PROGRAM craps(input,output);

(* THIS PROGRAM USES SEVERAL FUNCTIONS TO SIMULATE A GAME OF CRAPS.

IT INCLUDES A RANDOM NUMBER GENERATOR THAT IS SPECIFICALLY DESIGNED
FOR A COMPUTER THAT UTILIZES 16-BIT (2-BYTE) INTEGER QUANTITIES. *)

CONST seed = 12345;
VAR count,x : integer;
    n : real;
    answer : char;
    flag : boolean;

PROCEDURE play;
(* This procedure simulates a single game of craps *)
VAR score,tag : integer;

    FUNCTION throw : integer;
    (* This function simulates one throw of a pair of dice *)
    VAR d1,d2 : integer;

        FUNCTION rand : real;
        (* Note: This function can only be used with a computer that
           supports 16-bit (2-byte) integer quantities. *)
        VAR y,z : integer;
        BEGIN (* generate a random number *)
            z := 259*x;
            IF z >= 0 THEN y := z
            ELSE BEGIN y := z + 32767; y := y + 1 END;
            rand := 0.3051757E-4*y;
            x := y
        END; (* rand *)

    BEGIN (* throw the dice once *)
        d1 := 1 + trunc(6*rand);
        d2 := 1 + trunc(6*rand);
        throw := d1 + d2
    END; (* throw *)

    BEGIN (* play one game *)
        writeln;
        writeln(' Throw the dice . . . ');
        readln;
        score := throw;
        CASE score OF
            7,11 : BEGIN (* win on first throw *)
                write(score:3,' - Congratulations!');
                writeln(' You WIN on the first throw!');
                writeln
            END;
            2,3,12 : BEGIN (* loss on first throw *)
                write(score:3,' - Tough luck!');
                writeln(' You LOSE on the first throw!');
                writeln
            END;
        END;
    END;

```

(تكملة البرنامج في الصفحة التالية)

```

4,5,6,8,9,10 : BEGIN (* multiple throws *)
    tag := score;
    REPEAT
        write(score:3);
        writeln(' - Throw the dice again . . .');
        readln;
        score := throw;
    UNTIL (score = tag) OR (score = 7);
    IF score = tag
    THEN BEGIN
        write(score:3);
        write(' - You WIN by matching');
        writeln(' your first score');
        writeln
    END
    ELSE BEGIN
        write(score:3);
        write(' - You LOSE by failing to');
        writeln(' match your first score');
        writeln
    END
END (* multiple throws *)

END (* case *)
END; (* play *)

BEGIN (* executive routine *)
    x := seed;
    flag := true;
    page;
    writeln(' Welcome to the Game of CRAPSI');
    writeln;
    write(' Press the CARRIAGE RETURN to begin playing');
    readln;
    WHILE flag DO
        BEGIN
            play;
            write(' Do you want to play AGAIN? (Y/N) ');
            readln(answer);
            IF (answer = 'N') OR (answer = 'n') THEN flag := false
        END;
    writeln;
    writeln(' Bye, come back again!')
END.

```

يجب أن يذكر أن منتج الأعداد العشوائية ( الدالة rand ) مصمم لإنتاج سريان زائد لأرقام . وقد تكون هذه مشكلة لبعض المترجمات التي يمكن أن تنتج رسائل أخطاء ، وتفصل البرنامج دون إتمام تنفيذه إذا ماحدث مثل هذا السريان الزائد . وعادة مايمكن التغلب على هذه المشكلة بإضافة أمر خاص ، يجعل المترجم يتأكد من ضغط هذا السريان الزائد ، فإذا مانفذ هذا البرنامج - على سبيل المثال - على كمبيوتر شخصي من طراز IBM باستخدام مترجم بسكال IBM قياسي ( المعد من قبل Microsoft ) فيجب إضافة إحدى التعليمات ، وهي \$ MATHCK كتعليق . وعلى هذا ... تظهر rand كمايلي :

(انظر الصفحة التالية)



```

FUNCTION rand : real;
(*$MATHCK- *)
(* Note: This function can only be used with a computer that
   supports 16-bit (2-byte) integer quantities. *)
VAR y,z : integer;
BEGIN (* generate a random number *)
  z := 259*x;
  IF z >= 0 THEN y := z
    ELSE BEGIN y := z + 32767; y := y + 1 END;
  rand := 0.3051757E-4*y;
  x := y
END;

```

وتشغيل هذا البرنامج في وسط متداخل مثل الكمبيوتر الشخصى يعطى تعليمات كثيرة . وفيما يلي مجموعة مخرجات تقليدية من هذا البرنامج . وقد وضع خط تحت استجابة المستفيد .

```

Welcome to the Game of CRAPS!

Press the CARRIAGE RETURN to begin playing (press carriage return)

Throw the dice . . .

7 - Congratulations! You WIN on the first throw!

Do you want to play AGAIN? (Y/N) y

Throw the dice . . .

5 - Throw the dice again . . .

6 - Throw the dice again . . .

12 - Throw the dice again . . .

7 - You LOSE by failing to match your first score

Do you want to play AGAIN? (Y/N) y

Throw the dice . . .

9 - Throw the dice again . . .

4 - Throw the dice again . . .

9 - You WIN by matching your first score

Do you want to play AGAIN? (Y/N) y

Throw the dice . . .

2 - Tough luck! You LOSE on the first throw!

Do you want to play AGAIN? (Y/N) n

Bye, come back again!

```

وأخيرا يجب أن يكون القارئ حذرا من ناحية تغيير قيمة المتغير الشامل أو مؤشر المتغير داخل الدالة ، حيث إن هذا يمكن أن يؤدي إلى نتائج غير متوقعة وغير مطلوبة خارج الدالة . وأحيانا يشار إلى الأنشطة من هذا النوع بأنها تأثيرات جانبية Side effects للدوال . ( لاحظ أن مثل هذه التأثيرات الجانبية غير المطلوبة يمكن أن تنتج عن إجراء أيضا . والتأثيرات الجانبية عادة ماتكون أكثر خطورة مع الدوال - على أية حال - بسبب ميل المبرمجين إلى التفكير في الدالة بأنها تعود بقيمة واحدة فقط ) .

فمثلا الدالة rand في مثال ٧ - ١٨ تغير من قيمة مؤشر المتغير x في كل مرة تستدعى فيها الدالة ، وينتج عن ذلك تحديد قيمة مختلفة للمتغير x في الجزء الرئيسى للبرنامج . وهذا مطلوب طبيعياً لاعادة منتج الأعداد العشوائية . وبصفة عامة ... تجنب هذه العملية كلما كان ذلك ممكنا .

## ٥ - المزيد عن المؤشرات : 5. MORE ABOUT PARAMETERS

في بعض الأحيان يكون من المرغوب فيه أن يتصل إجراء معين أو دالة معينة بإجراء آخر ، أو بدالة أخرى يكون سبق تعريفه أو تعريفها ( توضيحه أو توضيحها ) خارج مدى الإجراء الذى يجرى الاتصال ، أو الدالة التى تجرى الاتصال . فقد نرغب - على سبيل المثال - فى جعل الإجراء A يستخدم الدالة B ، بالرغم من أن الدالة B معرفة خارج الإجراء A . ويمكن أن يتم ذلك عن طريق عبور الإجراء الخارجى أو الأدلة الخارجية ( أى الدالة B فى حالتنا هذه ) إلى إجراء معين أو دالة معينة ( أى الإجراء A فى حالتنا هذه ) كمؤشر . وعلى هذا ... تقدم لغة البسكال مؤشرات إجراء ومؤشرات دالة ، كما تقدم مؤشرات قيمة ومؤشرات متغير .

ويكتب توضيح المؤشر الرسمى لمؤشر إجراء أو لمؤشر دالة على هيئة عنوان إجراء ، أو عنوان دالة كما هو موضح أدناه .

مثال (٧-١٩)

افترض أن عملية الإجراء يجب أن تتصل بدالة حقيقية معرفة خارجيا ، وتحتوى على مؤشر من النوع الصحيح . دعنا نشير إلى هذه الدالة بأنها f . يمكن أن يبين الإجراء ( العملية ) على النحو التالى :

```
PROCEDURE process (FUNCTION f(u : integer) : real; c1,c2 : integer);
VAR c : integer;
    x : real;
BEGIN
  FOR c := c1 TO c2 DO
    BEGIN
      x := f(c);
      writeln(' x=',x)
    END
  END;
END;
```

ويمكن أن تحتوى المجموعة الرئيسة للبرنامج على الإشارة التالية لهذا الإجراء

```
process(calc,1,100);
```

حيث الدالة calc تكون معرفة داخل المجموعة الرئيسية . وفيما يلي التكوين الهيكلي لاحتويات المجموعة الرئيسية .

```
PROGRAM main(input,output);
.
.
FUNCTION calc(w : integer) : real;
.
.
BEGIN (* function calc *)
.
.
    calc := . . .
END; (* calc *)
PROCEDURE process (FUNCTION f(u : integer) : real; c1,c2 : integer);
VAR c : integer;
    x : real;
BEGIN
    FOR c := c1 TO c2 DO
        BEGIN
            x := f(c);
            writeln(' x=',x)
        END
    END;
END; (* process *)
.
.
BEGIN (* main block *)
.
.
    process(calc,1,100);
.
.
END.
```

يجب أن تتناظر مؤشرات الإجراءات الرسمية ، ومؤشرات الدالة الرسمية ، ومؤشرات الإجراءات الفعلية ، ومؤشرات الدالة الفعلية مؤشرات الخاصة . هذا التناظر يجب أن يحتوى على عدد المؤشرات وفتتها ونوعها .

وتكون مؤشرات الإجراءات ومؤشرات الدالة مفيدة بصفة خاصة عندما يتصل إجراء محدد ، أو دالة محددة بإجراءات أو دوال مختلفة different ( أى عند استخدام مؤشرات فعلية مختلفة ) عند نقاط استدعاء مختلفة واستخدام هذه الطريقة موضح فى المثالين التاليين .

### مثال (٧-٢٠)

اعتبر مرة أخرى عملية الإجراء المعرفة فى المثال السابق . يمكن الاتصال بهذا الإجراء عدة مرات فى المجموعة الرئيسية ، وذلك عن طريق مؤشرات دالة مختلفة موجودة فى كل اتصال . وعلى هذا ... قد تشمل المجموعة الرئيسية العبارات التالية :

```
process(calc,1,100);
```

```
.
```

```
.
```

```
process(flag,-10,50);
```

حيث الدالتان calc , flag معرفتان داخل المجموعة الرئيسية .

مثال (٧-٢١)

فيما يلي مثالاً لدالة تمر إلى دالة أخرى :

```
FUNCTION sum (FUNCTION f(x : real) : real; c1,c2 : integer) : real;
VAR c : integer;
    s : real;
BEGIN
    x := 0;
    FOR c := c1 TO c2 DO x := x + f(0.01*c);
    sum := x
END;
```

في هذا المثال تحسب الدالة sum مجموع العديد من القيم ، حيث تتحدد كل قيمة عن طريق الاتصال بالدالة f التي تمر إلى sum كمؤشر .

ويمكن أن تحتوي المجموعة الرئيسية على العديد من العبارات ، مثل مايلي :

```
value1 := sum(root(x),a,b);
.
.
.
value2 := sum(cube(x),1,n+1);
```

حيث cube و root هي دوال داخل المجموعة الرئيسية ، a , b , n هي متغيرات صحيحة النوع شاملة . لاحظ أن sum يقبل مؤشرات فعلية مختلفة ( أولاً root ثم cube ) في كل مرة يتم الاتصال به .

ولايسمح بسكال ISO القياسي بمرور الإجراءات القياسية أو الدوال الرئيسية كمؤشرات . وعلى أية حال ... فهناك بعض صيغ اللغة التي لاتضع هذا القيد .

مثال (٧-٢٢)

البحث عن أكبر قيمة . دعنا نعتبر مشكلة تعظيم التعبير التالي

$$y = x \cos (x)$$

مرة أخرى داخل المنطقة من  $x = 0$  وحتى  $x = \pi$  كما سبق ذكره في مثال ٧ - ١٤ دعنا نستخدم الآن دالة لتقويم هذا التعبير .

وفيما يلي صيغة لبرنامج تمر فيه الدالة max إلى الاجراء reduce كمؤشر .

```
PROGRAM maximum2(input,output);

(* THIS PROGRAM FINDS THE MAXIMUM OF A
   FUNCTION WITHIN A SPECIFIED INTERVAL *)

CONST sep = 0.0001;
VAR a,b,xl,xr,xmax,y1,yr,ymax : real;

FUNCTION curve(x : real) : real;
(* Define the function f=x*cos(x) *)
BEGIN
    curve := x*cos(x)
END;
```

(تكملة البرنامج في الصفحة التالية)

```

PROCEDURE reduce( FUNCTION f(x : real) : real;
                  VAR a,b,xl,xr,yl,yr : real);
(* Interval reduction routine *)
BEGIN
  xl := a + 0.5*(b-a-sep);
  xr := xl + sep;
  yl := f(xl);
  yr := f(xr);
  IF yl > yr THEN b := xr;  (* retain left interval *)
  IF yr > yl THEN a := xl  (* retain right interval *)
END;

BEGIN (* main action block *)
  readln(a,b);
  REPEAT
    reduce(curve,a,b,xl,xr,yl,yr);
  UNTIL (yl=yr) OR ( b-a <= 3*sep )
  xmax := 0.5*(xl+xr);
  ymax := curve(xmax);
  writeln(' xmax = ',xmax:8:6,'   ymax = ',ymax:8:6)
END.

```

وينتج عن تنفيذ هذا البرنامج نفس المخرجات الموضحة في مثال ٧ - ١٤ ويجب أن يكون مفهوماً - على أية حال - أنه ليس هناك ميزة خاصة في مرور الدالة إلى الإجراء في هذا المثال ، حيث إن الإجراء يستخدم نفس الدالة في كل مرة يتم الاتصال به . وعلى هذا ... تكون الطريقة الأكثر وضوحاً للاتصال بالدالة بدون تركها والذهاب إلى الإجراء كمؤشر . وفيما يلي صيغة لبرنامج يفعل ذلك .

```

PROGRAM maximum3(input,output);

(* THIS PROGRAM FINDS THE MAXIMUM OF A
   FUNCTION WITHIN A SPECIFIED INTERVAL *)

CONST sep = 0.0001;
VAR a,b,xl,xr,xmax,yl,yr,ymax : real;

FUNCTION curve(x : real) : real;
(* Define the function f=x*cos(x) *)
BEGIN
  curve := x*cos(x)
END;

PROCEDURE reduce( VAR a,b,xl,xr,yl,yr : real);
(* Interval reduction routine *)
BEGIN
  xl := a + 0.5*(b-a-sep);
  xr := xl + sep;
  yl := curve(xl);
  yr := curve(xr);
  IF yl > yr THEN b := xr;  (* retain left interval *)
  IF yr > yl THEN a := xl  (* retain right interval *)
END;

BEGIN (* main action block *)
  readln(a,b);
  REPEAT
    (تكملة البرنامج في الصفحة التالية)

```

```

      reduce(a,b,xl,xr,yl,yr)
UNTIL (yl=yr) OR ( b-a <= 3*sep );
xmax := 0.5*(xl+xr);
ymax := curve(xmax);
writeln(' xmax = ',xmax:8:6,'   ymax = ',ymax:8:6)
END.

```

وينتج عن تنفيذ هذا البرنامج النتائج الصحيحة المحسوبة المسماة :

xmax = 0.860586    ymax = 0.561096

لقد رأينا الآن ثلاث طرق مختلفة لحل هذه المشكلة . والسؤال أى هذه الطرق أفضل هو سؤال تعتمد اجابته على المبرمج ذلك بالرغم من أن آخر صيغة تبدو مناسبة لهذه المشكلة الخاصة .

## 6. RECURSION

## ٦ - الإعادة الذاتية :

إحدى السمات الجيدة للبرسكال هي امكانية استدعاء الإجراء نفسه ، وإمكانية استدعاء الدالة نفسها . وهذا مايعرف بالإعادة الذاتية . واستخدام الإعادة الذاتية مريح بصفة خاصة بالنسبة للمشاكل التى يمكن تعريفها باصطلاحات إعادة طيعية ( بالرغم من أن مثل هذه المشاكل يمكن أن تيرمج نون استخدام إعادة ذاتية أيضا ) .

عند كتابة إجراء إعادة ذاتية أو دالة إعادة ذاتية ، فمن الضرورى أن يحتوى هذا الإجراء ( أو هذه الدالة ) على شرط للإنهاء . وهذا يمنع الإعادة الذاتية من الاستمرار بصورة دائمة . وطالما أن شرط الإنهاء لم يتحقق ، فيقوم الإجراء ( أو الدالة ) باستدعاء نفسه فى المكان المناسب ، كما لو كان يستدعى أى إجراء آخر ، أو أى دالة أخرى .

مثال (٧-٢٢)

حساب المضروب . فى مثال ٧ - ١٧ رأينا برنامجا لحساب مضروب كمية مدخلات معطاه باستخدام دالة غير معادة ذاتيا فى أداء الحسابات الفعلية . دعنا نرى الآن كيف يمكن أداء هذه الحسابات باستخدام الإعادة الذاتية .

لاحظ أولا أنه يمكن تعريف مضروب كمية صحيحة موجبة  $n$  عن طريق الإعادة الذاتية بأنها

$$n! = n \times (n-1)!$$

حيث  $1! = 1$

وهذه المعادلات تقدم الأساس لدالة الإعادة الذاتية التالية :

```

FUNCTION factorial (n : integer) : integer;
(* Calculate the factorial of n *)
BEGIN
  IF n <= 1 THEN factorial := 1
  ELSE factorial := n*factorial(n-1)
END;

```

لاحظ أن هذه الدالة أبسط كثيراً عن الدالة المقدمة في مثال ٧ - ١٧. والتناظر القريب بين هذه الدالة وتعريف المشكلة الطبيعية بالنسبة لإعادة الذاتية يجب أن يكون ظاهراً. لاحظ أن جزء IF - THEN يقدم شرط إنهاء أيضاً ، والذي ينشط عندما يصبح المؤشر a أقل من أو يساوى 1 ( n لا يمكن أن تصبح أقل من 1 إلا إذا كانت قيمتها الأصلية أقل من 1 ) .

وفيما يلي برنامج بسكال كاملاً . والمجموعة الرئيسية تشبه جداً نظيرتها الموجودة في مثال ٧ - ١٧.

```
PROGRAM factorials3(input,output);

(* THIS PROGRAM USES A RECURSIVE FUNCTION TO CALCULATE
   THE FACTORIAL OF A GIVEN INTEGER QUANTITY *)

VAR x : integer;

FUNCTION factorial (n : integer) : integer;
(* Calculate the factorial of n *)
BEGIN
  IF n <= 1 THEN factorial := 1
  ELSE factorial := n*factorial(n-1)
END;

BEGIN (* main action block *)
  write(' Enter a positive integer: ');
  readln(x);
  writeln;
  writeln(' x= ',x,'    x! = ',factorial(x))
END.
```

وعلى هذا ... لحساب مضروب كمية فردية n ، يتم تكرار الاتصال بالدالة factorial مرة في المجموعة الرئيسية ، وعدد (n-1) من المرات داخل نفسها .

وفي بعض التطبيقات يسمح استخدام الإعادة الذاتية بتخزين عناصر بيانات متعددة داخل الكمبيوتر ، دون الحاجة إلى هيكل بيانات خاص . وهذا موضح في المثال التالي .

### مثال (٧-٢٤)

*الطباعة للخلف* . افترض أننا نريد كتابة برنامجاً يقرأ سطراً من أحد النصوص ، طوله غير محدد ، حتى يحدث نهاية للسطر ، وتعود العربية لبداية السطر الجديد . بعد ذلك يكتب السطر مطبوعاً بترتيب معكوس ( أى طباعة خلفية ) .

الطريقة التقليدية لهذه المشكلة هي عن طريق تخزين الرموز في قائمة كما قرأت ، ثم تطبع بترتيب عكسي ، وذلك بالاستمرار في القائمة من الخلف ، وذلك بعد الانتهاء من إدخال كل الرموز . ولأجل عمل ذلك ... يجب أن ننتج هيكل بيانات ( مثل المنظومة array ) لتخزين الرموز في الصورة التي أدخلت بها . ( سوف نرى كيف يمكن عمل ذلك في الفصل التاسع من الكتاب ) .

والطريقة الأخرى لحل هذه المشكلة هي استخدام الإعادة الذاتية ، كما هو موضح أدناه :

```
PROGRAM backwards(input,output);

(* THIS PROGRAM READS A LINE OF TEXT AND
   THEN WRITES IT OUT IN REVERSE ORDER. *)

PROCEDURE flipit;
(* Read single characters recursively,
   then write them out. *)
VAR c : char;

BEGIN
    read(c);
    IF NOT eoln THEN flipit;
    write(c)
END;

BEGIN (* main action block *)
    writeln(' Enter a line of text, then press RETURN');
    writeln;
    flipit
END.
```

والإجراء flipit هو مفتاح البرنامج . فيستدعى هذا الإجراء نفسه باستخدام الإعادة الذاتية ليقراً في الكمبيوتر رموزاً متتالية حتى ينتهي السطر ( حتى يتحقق eoln ) . لاحظ أن الاستدعاء للإعادة الذاتية يسبق عبارة write . وعلى هذا تقرأ كل الـ all الرموز داخل الكمبيوتر قبل طباعة أى منها خارجه . وبعد قراءة كل الرموز داخل الكمبيوتر . يكتب أحدث رمز تم دخوله خارج الكمبيوتر ( وذلك لأن هذا الرمز يناظر استدعاء للإجراء ) ، يليه الرمز السابق له مباشرة وهكذا .

فإذا كان البرنامج ينفذ - على سبيل المثال - بـ سطر المدخلات التالي :

NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR COUNTRY

فإن المخرجات المناظرة تصبح كمايلي :

YRTNUOC RIEHT FO DIA EHT OT EMOC OT NEM DOOG LLA ROF EMIT EHT SI WON

إذا ما استدعى إجراء ( أو دالة ) يحتوى على متغيرات محلية باستخدام الإعادة الذاتية ، فسوف تنتج فئة مختلفة different من المتغيرات المحلية أثناء كل مرة يحدث فيها استدعاء . وتكون أسماء المتغيرات المحلية بالطبع لها نفس الأسماء ( كما سبق توضيحها داخل الإجراء أو الدالة ) . وعلى أية حال ... فإن فئة مختلفة من القيم سوف تصاحب أسماء المتغيرات المحلية هذه فى كل مرة ينشط فيها الإجراء أو الدالة . وتصبح هذه القيم متاحة مع عدم اللف " unwinds " للإجراء أو الدالة ، كما تمت استدعاءات الإعادة الذاتية السابقة .

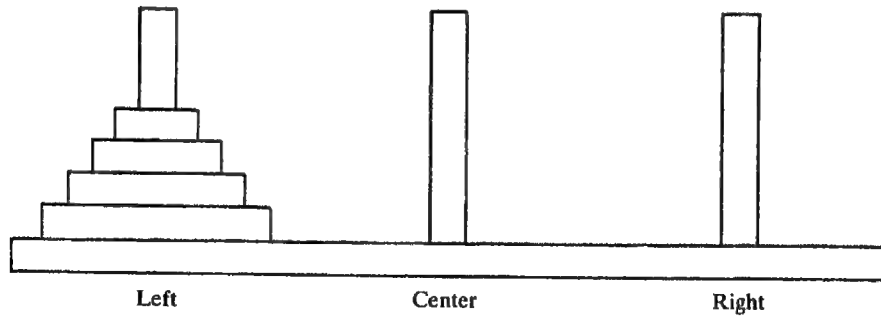
مثال (٧-٢٥)

اعتبر إجراء الإعادة الذاتية المسمى flipit الموضح فى مثال ٧ - ٢٤. يحتوى هذا الإجراء على متغير محلى من النوع الصحيح c . وفى كل مرة يستدعى الإجراء ، تتم قراءة رمز مختلف فى الكمبيوتر ويحدد للمتغير c . وعلى هذا ... يصاحب c العديد من الرموز المختلفة ، وتخزن على التوالى داخل ذاكرة الكمبيوتر . هذه الرموز تكون مستقلة عن بعضها . وتكتب خارج الكمبيوتر فى الترتيب المناسب عند إتمام استدعاءات الإعادة الذاتية ( والتي تعنى أن أحدث قيمة للمتغير c تم إدخالها وتخزينها وكتابتها خارج الكمبيوتر ) .



### مثال (٧-٢٦)

برج هانوى . فيما يلى مثالا يستخدم بكثرة لتوضيح كيفية استغلال الإعادة الذاتية فى تبسيط برمجة مشكلة تبدو معقدة . وتهتم المشكلة بلعبة أطفال شائعة الاستخدام ، تحتوى على ثلاثة أعمدة ، وعدد من الأقراص مختلفة الأحجام . وكل قرص به ثقب فى منتصفه لتركب حول الأعمدة . وفى البداية تتركب الأقراص على العمود الموجود فى أقصى اليسار ، مرتبة طبقا لحجمها ، على أن تكون الكبرى فى قاعدة العمود ، فالأقل فالأقل ، والصغرى فى نهايته ( انظر شكل ٧-٤ ) .



شكل ٧ - ٤

وهدف المباراة هو نقل الأقراص من العمود الموجود على أقصى اليسار إلى العمود الموجود على أقصى اليمين ، نون رفع قرص أكبر فوق قرص أصغر . ويتم تحريك قرص واحد فقط فى نفس الوقت . ويجب أن يوضع كل قرص فى أحد الأعمدة دائما .

والاستراتيجية العامة هى اعتبار أحد الأعمدة أنه الأصل ، وعمود آخر أنه الغاية . ويستخدم العمود الثالث فى التخزين المرحلى ليسمح بحركة الأقراص . وعلى هذا ... إذا وضع  $n$  قرص فى العمود الموجود على أقصى اليسار فى البداية ، فيمكن تمثيل مشكلة نقل هذه الأقراص التى عددها  $n$  إلى العمود الأيمن على النحو التالى :

(١) نقل أعلى (  $n-1$  ) قرص من العمود الأيسر إلى العمود المتوسط باستخدام العمود الأيمن كمخزن مرحلى .

(٢) نقل القرص المتبقى إلى العمود الأيمن .

(٣) نقل (  $n-1$  ) قرص الموجودة فى العمود المتوسط إلى العمود الأيمن باستخدام العمود الأيسر كمخزن مرحلى .

وعلى هذا ... يمكن تمثيل المشكلة فى صورة الإعادة الذاتية المطبقة مع أى قيمة لـ  $n$  أكبر من 0 . ( عندما تكون  $n=1$  ، فإننا ننقل قرصاً ببساطة من العمود الأيسر إلى العمود الأيمن ) .

ولكى نكتب برنامجا لهذه المباراة ، فإننا نضع أرقاما أولا للأعمدة ، بحيث إن  $left = 1$  و  $center = 2$  و  $right = 3$  ، ثم نعد بعد ذلك إجراء إعادة ذاتية يسمى  $transfer$  ، والذي ينقل  $n$  قرصا من عمود إلى عمود آخر . ونشير إلى الأعمدة بمتغيرات صحيحة النوع اسمها  $origin$  ،  $destination$  ،  $other$  . وعلى هذا ... فإذا ما حددنا القيمة 1 للمتغير  $origin$  والقيمة 3 للمتغير  $destination$  والقيمة 2 للمتغير  $other$  ، فإننا نكون فى واقع الأمر محددين حركة  $n$  قرص من العمود 1 ( العمود الموجود على أقصى اليسار ) إلى العمود 3 ( العمود الموجود على أقصى اليمين ) باستخدام العمود 2 ( العمود الموجود فى المنتصف ) فى التخزين المرحلى . وبهذه الملاحظات يكون للإجراء الهيكل التالى :

```
PROCEDURE transfer(n,origin,destination,other : integer);
IF n>0 THEN
  (* transfer n-1 disks from the origin to the intermediate pole *)
  (* transfer the remaining disk from the origin to the destination *)
  (* transfer n-1 disks from the intermediate pole to the destination *)
END;
```

يمكن حدوث نقل (n-1) قرص عن طريق استدعاء الإعادة الذاتية لـ transfer . وعلى هذا ... يمكننا كتابة .

```
transfer(n-1,origin,other,destination)
```

لأول نقل وكتابة

```
transfer(n-1,other,destination,origin)
```

لنقل الثاني . ( لاحظ ترتيب المؤشرات في كل استدعاء ) . ويتطلب نقل قرص واحد من الأصل إلى الغاية كتابة قيم destination , origin . ويمكن أن يحدث ذلك باستدعاء إجراء آخر اسمه diskmove . وهو معرف داخل transfer ويمكن كتابة إجراء النقل كاملاً كما يلي :

```
PROCEDURE transfer(n,origin,destination,other : integer);

  PROCEDURE diskmove(origin,destination : integer);
  BEGIN
    writeln('Move ',origin:1,' to ',destination:1)
  END;

  BEGIN
    IF n>0 THEN BEGIN
      transfer(n-1,origin,other,destination);
      diskmove(origin,destination);
      transfer(n-1,other,destination,origin)
    END
  END;
```

وأصبح الأمر الآن سهلاً ، وذلك بإضافة المجموعة الرئيسية التي تقرأ أساساً قيمة n ، ثم تبدأ الحسابات باستدعاء transfer . وفي أول مرة لاستدعاء الإجراء تحدد المؤشرات الفعلية بأنها أرقام صحيحة . أى أن

```
transfer(n,1,3,2)
```

وهذا يحدد نقل كل الأقراص التي عددها n من العمود 1 ( عمود الأصل ) إلى العمود 3 ( عمود الغاية ) باستخدام عمود 2 في التخزين المرحلي .

وفيما يلي البرنامج كاملاً :

```

PROGRAM towersofhanoi(input,output);

(* THIS PROGRAM SOLVES A WELL-KNOWN GAME
   USING RECURSIVE PROCEDURE CALLS *)

VAR n : integer;

PROCEDURE transfer(n,origin,destination,other : integer);
(* Transfer n disks from the origin to the destination *)

    PROCEDURE diskmove(origin,destination : integer);
    (* Move a single disk from the origin to the destination *)
    BEGIN
        writeln('Move ',origin:1,' to ',destination:1)
    END; (* diskmove *)

    BEGIN (* transfer *)
        IF n>0 THEN BEGIN
            transfer(n-1,origin,other,destination);
            diskmove(origin,destination);
            transfer(n-1,other,destination,origin)
        END
    END; (* transfer *)

BEGIN (* main action block *)
    write(' Enter the number of disks -> ');
    readln(n);
    writeln;
    transfer(n,1,3,2)
END.

```

ويجب أن يكون مفهوماً أن مجموعات القيم المختلفة different تعرف لمؤشرات القيمة other , destination , origin , n في كل مرة تستدعى فيها transfer . هذه المجموعات من القيم تخزن منفصلة عن بعضها داخل ذاكرة الكمبيوتر . وهذه إمكانية في تخزين هذه المجموعات المنفصلة من القيم ، ثم استعادتها في الوقت المناسب ، هي التي تسمح بعمل إعادة ذاتية .

وعند تنفيذ البرنامج لقيمة  $n = 3$  نحصل على المخرجات التالية . ( لاحظ أن استجابة المستفيد موضوع تحتها خط ) .

Enter the number of disks -> 3

Move 1 to 3  
 Move 1 to 2  
 Move 3 to 2  
 Move 1 to 3  
 Move 2 to 1  
 Move 2 to 3  
 Move 1 to 3

ويجب أن يفكر القارئ في هذا التسلسل للنقل للتحقق من صحته .

وعلى القارئ دراسة هذا المثال بعناية ، فالمنطق ليس سهلا ، بالرغم من بساطة البرنامج .

في كل أمثلة الإعادة الذاتية التي رأيناها حتى الآن يقوم إجراء واحد ( أو دالة واحدة ) باستدعاء نفسه . وهناك صيغة أخرى للإعادة الذاتية تسمى إعادة ذاتية مشتركة ، والتي يستدعى إجراء ( أو دالة ) إجراء آخر ، وبعد ذلك يستدعى الإجراء الثاني ( أو الدالة ) الإجراء الأول . تذكر على أية حال أنه لا يمكن الاتصال بإجراء ( أو بدالة ) قبل توضيحه . وعلى هذا ... فهناك مشكلة في الإعادة الذاتية المشتركة عندما يستدعى الإجراء ( الدالة ) الأول الإجراء الثاني قبل أن يكون معرفا .

ويحل البسكال هذه المشكلة بالسماح بتوضيح صوري dummy declaration للإجراء الثاني ( أو الدالة الثانية ) يسبق الإجراء الأول ( أو الدالة الأولى ) . يحتوى التوضيح الصوري على إجراء ( أو دالة ) بداية يتبعها الكلمة الأساسية forward . أما التوضيح الحقيقي للإجراء ( أو الدالة ) ، فيكتب بعد ذلك في المكان المناسب بعد توضيح أول إجراء ( أو دالة ) . ويكتب بالطريقة المعتادة ، فيما عدا أن قائمة المؤشر الرسمي تحذف من العنوان . ( وفي حالة الدالة يحذف نوع النتيجة أيضا ) .

مثال (٧-٢٧)

فيما يلي تخطيطاً هيكلياً لدالة تستدعى إجراء ، ثم بعد ذلك تستدعى الدالة باستخدام الإعادة الذاتية .

```
FUNCTION second(a,b : real) : real; forward;

PROCEDURE first(x,y : real);
VAR w : real;
BEGIN
    .
    .
    w := second(x,y);
    .
    .
END;

FUNCTION second;
VAR u,v : real;
BEGIN
    .
    .
    first(u,v);
    .
    .
    second := . . .
END;
```

لاحظ أن التوضيح الصوري للمتغير second ضروري ، بحيث إن first يمكنه الاتصال بـ second قبل أن يتم توضيح second . وبعد ذلك فإن التوضيح الحقيقي لـ second يلي توضيح first ، لاحظ - على أية حال - أن عنوان الدالة مختصر الآن .

وأخيرا يجب أن يكون مفهوما أن استخدام الإعادة الذاتية ليس ضروريا أن يكون أفضل طريقة لحل المشكلة ، بالرغم من أن تعريف المشكلة قد يكون له طبيعة إعادة ذاتية . فالحل دون استخدام الإعادة الذاتية قد يكون أكثر كفاءة بالنسبة لاستغلال ذاكرة الكمبيوتر وسرعة التنفيذ . وعلى هذا ... فإن تكرار استخدام الإعادة الذاتية يشتمل على الموازنة بين البساطة وسوء الأداء . ويجب الحكم على ذلك على كل مشكلة طبقا لمزاياها الخاصة لها .

## Review Questions

## أسئلة للمراجعة :

- (١) ماهى مميزات تجزئة البرنامج ؟ ماهى أنواع أجزاء البرنامج المتاحة فى البسكال ؟
- (٢) كيف يمكن الإشارة إلى إجراء ؟ وماهى الكلمات الأخرى المستخدمة فى الإشارة إلى دليل إجراء ؟
- (٣) صف هيكل قواعد إعداد الإجراء . كيف يكتب عنوان الإجراء ؟
- (٤) فى أى مكان من برنامج البسكال يجب أن يظهر توضيح الإجراء ؟
- (٥) ماهو الفرق بين المعارف الشاملة والمعارف المحلية ؟ وأيها يفضل استخدامه ، ولماذا ؟
- (٦) ماذا يعنى مدى المعارف ؟
- (٧) ماذا تعنى الإجراءات المتداخلة ؟ ماهى القيود التى تتطلب اهتماما خاصا عند محاولة الاتصال بإجراءات متداخلة ؟
- (٨) هل مفهوم المدى يطبق على المعارف التى تمثل ثوابت ومتغيرات فقط ؟ وضح ذلك .
- (٩) كيف تؤثر قواعد المدى على نقل التحكم الذى يصاحب استخدام عبارة GOTO ؟
- (١٠) ماهى المؤشرات ؟ وكيف تستخدم ؟
- (١١) ماهو الفرق بين المؤشرات الفعلية والمؤشرات الرسمية ؟
- (١٢) لخص القواعد التى يجب اتباعها عند التعويض بمؤشرات فعلية ، بدلا من مؤشرات رسمية .
- (١٣) ماهى الأربع فئات الخاصة بالمؤشرات الرسمية ؟ وكيف تختلف عن بعضها ؟
- (١٤) ماهى مؤشرات القيمة ؟ كيف تكتب مؤشرات القيمة الرسمية ؟ وفى أى نوع من أنواع التطبيقات تستخدم مؤشرات القيمة بصفة عامة ؟
- (١٥) ماهى مؤشرات المتغير ؟ كيف تكتب مؤشرات المتغير الرسمية ؟ وكيف تختلف مؤشرات المتغير عن مؤشرات القيمة بالنسبة لـ :
- (أ) طريقة كتابتها .
- (ب) طريقة استخدامها .
- (١٦) ماهى الدالة ؟ كيف تختلف الدالة عن الإجراء ؟
- (١٧) كيف يشار إلى الدالة ؟ قارن ذلك بالإشارة إلى الإجراء .
- (١٨) صف هيكل قواعد إعداد الدالة . كيف يكتب عنوان الدالة ؟ وكيف يختلف هيكل القواعد هذا عن نظيره للإجراء ؟

(١٩) أين يجب أن يظهر توضيح الدالة في برنامج البسكال ؟ وهل هناك ترتيب خاص يجب اتباعه بالنسبة لتوضيحات الدوال وتوضيحات الإجراءات ؟

(٢٠) هل يمكن أن تتداخل الدوال بنفس الطريقة مثل الإجراءات ؟ وهل يمكن تبادل تداخل الإجراءات والدوال ؟

(٢١) ماذا تعني الآثار الجانبية التي تصاحب الدالة ؟ وهل يصاحب الإجراءات آثار جانبية أيضا ؟

(٢٢) متى يكون مرغوبا في مرور الإجراء أو الدالة كمؤشر ؟

(٢٣) متى يمر الإجراء أو الدالة كمؤشر ، وماذا يجب أن يحفظ بين المؤشرات الرسمية والمؤشرات الفعلية ؟

(٢٤) كيف تكتب مؤشرات الإجراء الرسمية ومؤشرات الدالة الرسمية ؟

(٢٥) هل يمكن أن يمر الإجراء القياسي أو الدالة القياسية كمؤشر ؟

(٢٦) ماذا تعني إعادة الذاتية ؟ وهل هي أساسية في البرمجة ؟

(٢٧) متى يكتب إجراء أو دالة إعادة ذاتية ، وكيف يمكن منع عملية إعادة الذاتية من الاستمرار اللانهائي ؟

(٢٨) ماهي المميزات التي تقدمها إعادة الذاتية بالنسبة لتخزين عناصر بيانات متعددة ؟

(٢٩) افترض أن إجراء ( أو دالة ) إعادة ذاتية يحتوى على مجموعة من المتغيرات المحلية . كيف تفسر المتغيرات المحلية كلما تكرر الاتصال بالإجراء ( أو الدالة ) .

(٣٠) ماذا تعني إعادة الذاتية المشتركة ؟

(٣١) متى يعرف إجراء ( أو دالة ) إعادة ذاتية مشتركة ، وماهي المشكلة التي يحلها التوضيح الصوري ؟ وكيف تكتب التوضيحات الصورية ؟ وماهي التغييرات التي يجب إجراؤها على التوضيحات الأخرى بسبب التوضيحات الصورية ؟

## Solved Problems

## مسائل محلولة :

(٣٢) فيما يلي العديد من تخطيطات دلائل إجراءات ودوال . ( افترض أن sample1 , sample2 , sample هي إجراءات ، وأن demo1 , demo2 , demo هي دوال . التوافقية في النوع مفروضة خلال التخطيطات ) .

- |                              |  |
|------------------------------|--|
| (a) sample;                  | (f) z := a + 2*demo - 3;                             |
| (b) sample1(a,b,c);          | (g) z := demo1(a,b,c);                               |
| (c) sample1(a+b,2*x,sqr(c)); | (h) z := y + demo1(a+b,2*x,sqr(c));                  |
| (d) sample2(sample,demo);    | (i) z := demo2(sample,demo);                         |
| (e) z := demo;               | (j) IF demo1(a,b,c) < zstar THEN z := demo1(a,b,c);; |

(٣٣) فيما يلي العديد من هياكل توضيحات إجراءات ودوال . ( هذه التوضيحات تناظر الإجراءات والدوال المشار إليها في المشكلة رقم ٣٢ ) .

```
(a) PROCEDURE sample;
    VAR . . .; (* local variables *)
    BEGIN
```

```
    .
    .
    .
```

```
    END;
```

```
(b) PROCEDURE sample1(x,y,z : real);
    VAR . . .; (* local variables *)
    BEGIN
```

```
    .
    .
    .
```

```
    END;
```

```
(c) PROCEDURE sample1(VAR x,y,z : real);
    VAR . . .; (* local variables *)
    BEGIN
```

```
    .
    .
    .
```

```
    END;
```

( لاحظ أن الجزء (b) يستخدم مؤشرات قيمة ، بينما يستخدم الجزء (c) مؤشرات متغير . مؤشرات المتغير ضرورية إذا ما تغيرت المؤشرات داخل الإجراء ، وعادت القيم الجديدة إلى نقاط الاتصال ) .

```
(d) PROCEDURE sample2(PROCEDURE sample; FUNCTION demo : real);
    VAR . . .; (* local variables *)
    BEGIN
```

```
    .
    .
    .
```

```
    END;
```

```
(e) PROCEDURE sample2(PROCEDURE sample1(x,y,z : real);
    FUNCTION demo1(VAR a,b,c : real) : real);
    VAR . . .; (* local variables *)
    BEGIN
```

```
    .
    .
    .
```

```
    END;
```

```
(f) FUNCTION demo : real;
    VAR . . .; (* local variables *)
    BEGIN
```

```
    .
    .
    .
```

```
    demo := . . .
```

```
    END;
```

```
(g) FUNCTION demo1(x,y,z : real) : real;
    VAR . . .; (* local variables *)
    BEGIN
```

```
    .
    .
    .
```

```
    demo1 := . . .
```

```
    END;
```

```
(h) FUNCTION demo2(PROCEDURE sample; FUNCTION demo : real) : real;
    VAR . . .; (* local variables *)
    BEGIN
        .
        .
        .
        demo2 := . . .
    END;

(i) FUNCTION demo2(PROCEDURE sample1(VAR x,y,z : real);
    FUNCTION demo1(a,b,c : real) : real;
    VAR . . .; (* local variables *)
    BEGIN
        .
        .
        .
        demo2 := . . .
    END;
```

(٣٤) الهيكل التالي هو تخطيط يبين توضيح إجراء (أو دالة) مع الأدلة المناظرة له (لها) .

```
(a) PROGRAM sample1(input,output);
    VAR a,b,c : integer;

    PROCEDURE proc1(a,b,c : integer);
    VAR . . .; (* local variables *)
    BEGIN
        .
        .
        .
    END;

    FUNCTION funct1(x,y : integer) : integer;
    VAR r,s,t : integer;
    BEGIN
        .
        .
        .
        proc1(r,s,t);
        .
        .
        funct1 := . . .
    END;

    BEGIN (* main action statements *)
        .
        .
        c := funct1(3,-1);
        .
        .
        proc1(a,b+c,0);
        .
        .
        proc1(a+b,funct1(0,1),c+3)
    END.
```



```

(b) PROGRAM sample2(input,output);
    VAR a,b,c : integer;

    FUNCTION funct2(x,y : integer) : integer; forward;

    PROCEDURE proc2(a,b,c : integer);
    VAR p,q,r : integer;
    BEGIN
        .
        .
        r := funct2(p,q-1);
        .
        .
    END;

    FUNCTION funct2;
    VAR r,s,t : integer;
    BEGIN
        .
        .
        proc2(r,s,t);
        .
        .
        funct2 := . . .
    END;

    BEGIN (* main action statements *)
        .
        .
        c := funct2(3,-1);
        .
        .
        proc2(a,b+c,0);
        .
        .
        proc2(abs(a+b),c,funct2(1,1))
    END.

```

لاحظ أن هذه المثال يستخدم سمة التقدم forward .

```

(c) PROGRAM sample3(input,output);
    VAR a,b,c : integer;

    FUNCTION funct3(p,q : integer) : integer;
    VAR r,s,t : integer;

    PROCEDURE proc3(VAR x,y,z : integer);
    VAR . . .; (* local variables *)
    BEGIN
        .
        .
        .
    END;

```

```

BEGIN (* funct3 action statements *)
.
.
proc3(r,s,t);
.
.
funct3 := . . .
END;

BEGIN (* main action statements *)
.
.
c := funct3(1,2);
.
.
a := funct3(2,0);
.
.
END;

```

Note that proc3 is embedded within funct3.

```

(d) PROCEDURE factorial (VAR f : integer; n : integer);
BEGIN
  IF n <= 1 THEN f := 1
  ELSE BEGIN
    factorial (f,n-1);
    f := n*f
  END
END;

```

يوضح هذا المثال استخدام إعادة لإجراء ( قارن ذلك مع الدالة المبينة في مثال ٧ - ٣٢. لاحظ أن هذا الإجراء يستخدم كل من مؤشر متغير (f) ومؤشر قيمة (a) .

## Supplementary Problems

## مشاكل متكاملة :

(٣٥) التخطيطات الهيكلية التالية توضح حالات مختلفة تحتوى على استخدام إجراءات ووال . بعضها مكتوب بصورة غير صحيحة . عرف الأخطاء .

```

(a) PROCEDURE sum(a,b,c : real);
  VAR sum : real;
  BEGIN
    sum := a + b + c;
  END;

(b) FUNCTION sum(a,b,c : real) : real;
  BEGIN
    sum := a + b + c
  END;

(c) PROCEDURE sum1(a,b,c : real; VAR sum : real),
  BEGIN
    sum := a + b + c
  END;

```

---

```

(d) FUNCTION sum(x : real);
    VAR sum : real;
    BEGIN
        sum := sum + x
    END;

(e) PROGRAM sample(input,output);
    VAR a,b,c : char;

        PROCEDURE procl(x,y : integer);
        BEGIN
            .
            .
            .
        END;

    BEGIN (* main action statements *)
        .
        .
        procl(a,b);
        .
        .
    END.

(f) PROGRAM sample(input,output);
    VAR a,b : integer;
        flag : boolean;

        PROCEDURE procl(x,y : integer);
        VAR . . .; (* local variables *)
        BEGIN
            .
            .
            REPEAT
                .
                .
                IF . . . THEN flag := false;
                .
                .
            UNTIL flag = false
        END;

    BEGIN (* main action statements *)
        .
        .
        flag := true;
        procl(a,a+2);
        .
        .
        flag := true;
        procl(abs(b-a),0);
        .
        .
    END.

(g) PROGRAM sample(input,output);
    VAR a,b,t : integer;

        PROCEDURE proc2(VAR x,y : integer);
        VAR . . .; (* local variables *)

```

```

BEGIN
.
.
IF t > 0 THEN y := . . .
ELSE y := . . .;
.
.
END;
BEGIN (* main action statements *)
.
.
readln(n);
.
.
proc2(2*a+1,b);
.
.
proc2(abs(a-10),b);
.
.
END.
(h) PROGRAM sample(input,output);
VAR a,b,c : char;
    t1,t2,t3 : boolean;

    FUNCTION upper(x : char) : boolean;
    BEGIN
        IF ord(x) <= 90 THEN upper := true
        ELSE upper := false
    END;

    BEGIN (* main action statements *)
        .
        .
        t1 := upper(a);
        .
        .
        t2 := upper(b,c);
        .
        .
        t3 := upper(a+b)
    END.
(i) PROGRAM sample(input,output);
VAR a,b : integer;

    FUNCTION smaller : integer;
    BEGIN
        IF a < b THEN smaller := a
        ELSE smaller := b
    END;

    BEGIN (* main action statements *)
        .
        .
        writeln(' Smallest number: ',smaller);
        .
        .
    END.

```

---

```

(j) PROGRAM sample(input,output);
    VAR a,b : integer;

    PROCEDURE procl(x,y : integer);
    VAR a,b : char;
    BEGIN
        .
        .
        .
    END;

    BEGIN (* main action statements *)
        .
        .
        procl(a,b);
        .
        .
    END.

(k) PROGRAM sample(input,output);
    VAR a,b,c : integer;
        x,y : real;

    FUNCTION funct1(x,y : integer) : real;
    VAR . . .; (* local variables *)
    BEGIN
        .
        .
        c := . . .;
        .
        .
        funct1 := . . .
    END;

    BEGIN (* main action statements *)
        .
        .
        x := funct1(a,b);
        .
        .
        y := funct1(b,a-2);
        .
        .
    END.

(l) PROGRAM sample(input,output);
    VAR x,y,z : real;

    FUNCTION smaller(FUNCTION f1(x : real) : real;
                     FUNCTION f2(x : real) : real) : real;
    BEGIN
        IF f1(x) < f2(x) THEN smaller := f1(x)
        ELSE smaller := f2(x)
    END;

```

```

BEGIN  (* main action statements *)
.
.
y := smaller(sin,cos);
.
.
z := smaller(log,sqrt);
.
.
END.

(m) PROGRAM sample(input,output);
VAR a,b,c : integer;

    FUNCTION funct1(x,y : integer) : integer;
    VAR . . .;  (* local variables *)
    BEGIN
        .
        .
        .
        funct1 := . . .
    END;

    PROCEDURE procl(a,b,c : integer);
    VAR r,s,t : integer;
    BEGIN
        .
        .
        r := funct1(s,t);
        .
        .
    END;

BEGIN  (* main action statements *)
.
.
c := funct1(3,-1);
.
.
procl(a,b+c,0);
.
.
procl(a+b,funct1(0,1),c+3);
.
.
END.

(n) PROGRAM sample(input,output);
VAR a,b,c : integer;

    PROCEDURE procl(a,b,c : integer);
    VAR r,s,t : integer;
    BEGIN
        .
        .
        r := funct1(s,t);
        .
        .
    END;

```

```

        FUNCTION funct1(x,y : integer) : integer;
        VAR . . .;  (* local variables *)
        BEGIN
            .
            .
            .
            funct1 := . . .
        END;
    BEGIN  (* main action statements *)
        .
        .
        c := funct1(3,-1);
        .
        .
        proc1(a,b+c,0);
        .
        .
        proc1(a+b,funct1(0,1),c+3);
        .
        .
    END.
(o) PROGRAM sample(input,output);
    VAR a,b,c : integer;

    PROCEDURE test(VAR : x,y,z : integer);
    VAR r,s,t : integer;

        FUNCTION value(p,q : integer) : integer;
        VAR . . .;  (* local variables *)
        BEGIN
            .
            .
            .
            value := . . .
        END;
    BEGIN  (* test action statements *)
        .
        .
        r := value(s,t);
        .
        .
    END;

    BEGIN  (* main action statements *)
        .
        .
        test(a,b,c);
        .
        .
        c := value(a,b);
        .
        .
    END.

```

```
(p) FUNCTION factorial(n : integer) : integer;
    BEGIN
        factorial := n*factorial(n-1)
    END;
```

(٣٦) عبر عن كل من الصيغ التالية بصيغة الاعداد الذاتية :

- (a)  $y = (x_1 + x_2 + \dots + x_n)$
- (b)  $y = 1 - x + x^2/2 - x^3/6 + x^4/24 + \dots + (-1)^n x^n/n!$
- (c)  $p = (f_1 * f_2 * \dots * f_i)$

## Programming Problems

مشاكل مبرمجة :

(٣٧) اكتب إجراء يحسب الجذور الحقيقية للمعادلة :

$$ax^2 + bx + c = 0$$

مستخدما العلاقة :

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



افرض أن  $a, b, c$  هي مؤشرات قيمتها معطاه ، وأن  $x_1, x_2$  هي الجذور المحسوبة . افرض أيضا أن  $b^2 > 4ac$  بحيث إن الجذور المحسوبة تكون دائما حقيقية ومميزه .

(٣٨) اكتب برنامجا كاملا بلغة البسكال يحسب الجذور الحقيقية لمعادلة .

$$ax^2 + bx + c = 0$$

مستخدما صيغة حساب الجذر المعطاه في المشكلة رقم ٧ - ٣٧ . اقرأ المعاملات  $a, b, c$  داخل الكمبيوتر في المجموعة الرئيسية للبرنامج . ثم اتصل بالإجراء المكتوب في المشكلة السابقة للحصول على الحل المطلوب . اكتب في النهاية قيم المعاملات ، يتبعها قيم  $x_1, x_2$  المحسوبة . تأكد أن كل المخرجات لها عناوين واضحة .

$$\begin{array}{ccc} a=2 & b=6 & c=1 \\ 3 & 3 & 0 \\ 1 & 3 & 1 \end{array}$$

(٣٩) عدل الإجراء المكتوب في المشكلة ٣٧ - ٧ ، بحيث إن كل جنور المعادلة .

$$ax^2 + bx + c = 0$$

تحسب مع معرفة  $a, b, c$  . لاحظ أن الجذور سوف تتكرر ( أى أنه سيكون هناك جذر حقيقى واحد فقط ) إذا ماكانت  $b^2 = 4ac$  ، كما أن الجذور تكون مركبة أيضا إذا ماكان  $b^2 < 4ac$  ، فى هذه الحالة يتحدد الجزء الحقيقى للكسر بأنه :

$$-b/2a$$

ويتحدد الجزء التخيلى للكسر بأنه :

$$(\pm\sqrt{4ac - b^2})i$$

حيث  $i$  تمثل  $\sqrt{-1}$

(٤٠) عدل برنامج البسكال المكتوب في المشكلة رقم ٧ - ٣٩ . تأكد أن كل المخرجات لها عناوين واضحة . اختبر البرنامج مستخدما البيانات التالية .

$$\begin{array}{ccc} a=2 & b=6 & c=1 \\ 3 & 3 & 0 \\ 1 & 3 & 1 \\ 0 & 12 & -3 \\ 3 & 6 & 3 \\ 2 & -4 & 3 \end{array}$$

(٤١) اكتب دالة تسمح برفع عدد حقيقى أو صحيح إلى قوة صحيحة . وفى كلمات أخرى ... نحن نرغب فى تقويم الصيغة :

$$y = x^n$$

حيث  $x, y$  متغيران من النوع الصحيح أو الحقيقي ، و  $n$  متغير من النوع الصحيح . ( لاحظ أن  $x, y$  لابد أن يكونا من نفس النوع دائما )

(٤٢) اكتب برنامجا كاملا بلغة البسكال يقرأ قيما عددية لكل من  $x, n$  لتقويم الصيغة الموجودة في المشكلة رقم ٤١ - ٧ .

$$y = x^n$$

مستخدما الدالة المكتوبة في المشكلة رقم ٧ - ٤١ ، ثم اخرج النتائج المحسوبة . اختبر البرنامج مستخدما البيانات التالية :

$x =$	$n =$
2	3
2	12
2	-5
-3	3
-3	7
-3	-5
1.5	3
1.5	10
1.5	-5
0.2	3
0.2	5
0.2	-5

(٤٣) وسع الدالة المكتوبة في المشكلة رقم ٧ - ٤١ بحيث يمكن رفع  $x$  لأي قوة ، سواء أكانت حقيقية أم صحيحة ( ملاحظة : استخدم اللوغاريتم إذا كان الأس من النوع الحقيقي . الصيغة هي  $y = x^n = e^{(n \ln x)}$  تذكر أن تختبر القيم السالبة لـ  $x$  ) . حدد هذه الدالة في البرنامج المكتوب في المشكلة رقم ٧ - ٤٢ . اختبر البرنامج مستخدما البيانات المعطاه في المشكلة رقم ٧ - ٤٢ مع البيانات الإضافية التالية .

$x =$	$n =$
2	0.2
2	-0.8
-3	0.2
-3	-0.8
1.5	0.2
1.5	-0.8
0.2	0.2
0.2	-0.8
0.2	0.0

(٤٤) أعد برنامج شفرة الرموز المعطى في مثال ٦ - ٣١ بحيث إنه يستخدم إجراء . ترجم ونفذ البرنامج للتأكد من أنه يعمل بطريقة صحيحة .

(٤٥) عدل البرنامج لحساب حل معادلة جبرية معطاه في مثال ٦ - ١٤ بحيث إن كل تكرار يحدث داخل دالة . ترجم ونفذ البرنامج للتأكد من أنه يعمل بطريقة صحيحة .

(٤٦) عدل برنامج حساب متوسط قائمة بالأعداد المعطاه فى مثال ٧ - ٨ ، بحيث إنه يستخدم دالة . اختبر البرنامج مستخدما العشرة أعداد التالية :

27.5	87.0
13.4	39.9
53.8	47.7
29.2	8.1
74.5	63.2

(٤٧) يمكن أن يعدل البرنامج المعطى فى مثال ٧ - ١٤ ببساطة لتقليل ( لتدنية ) minimize دالة فى  $x$  . مثل هذه العملية للتقليل ( للتدنية ) يمكن أن توفر لنا أسلوب مرتفع الكفاءة لحساب جذور معادلة جبرية غير خطية . افرض مثلا أننا نريد إيجاد قيمة خاصة لـ  $x$  تتسبب فى أن الدالة  $f(x) = 0$  . الدالة التقليدية من هذا النوع يمكن أن تأخذ الصورة .

$$f(x) = x + \cos(x) - 1 - \sin(x).$$

إذا جعلنا  $g(x) = f(x)^2$  ، فتكون الدالة  $g(x)$  موجبة دائما ، إلا إذا كانت قيم  $x$  هى جذور الدالة المعطاه ، أى للدالة  $f(x)$  ، وبالتالي  $g(x)$  ، والتي تساوى صفرا . وعلى هذا ... فأى قيمة لـ  $x$  تدنى  $g(x)$  تكون جذرا للمعادلة  $f(x) = 0$  أيضا . عدل البرنامج الموضح فى مثال ٧ - ١٤ لتدنية دالة معطاه . استخدم البرنامج فى الحصول على جذور المعادلات التالية :

$$(a) \quad x + \cos(x) = 1 + \sin(x), \quad \pi/2 < x < \pi$$

$$(b) \quad x^5 + 3x^2 = 10, \quad 0 \leq x \leq 3$$

(٤٨) عدل البرنامج الموضح فى مثال ٧ - ١٨ ، بحيث يحاكي تسلسل المباراة تلقائيا بطريقة غير متكررة . حدد أعدادا من النوع الصحيح يحدد إجمالى عدد مرات المكسب ومتغير مدخلات صحيح قيمته تحدد عدد المباريات التي تحاكي .

استخدم البرنامج فى محاكاة عدد كبير من المباريات ( أى 1000 مباراة مثلا ) . قدر احتمال المكسب عند لعب المباراة . ( هذه القيمة 6 معبر عنها ككسر عشري 6 تساوى عدد مرات المكسب مقسوما على إجمالى عدد المباريات الملعوبة . إذا تعدى الاحتمال 0.50 فهذا فى صالح اللاعب ) .

(٤٩) أعد كتابة كل من البرامج التالية ، بحيث إنها تشمل إجراء واحد على الأقل ، أو دالة واحدة على الأقل . كن صريحا عند اختيارك المؤشرات .

- حساب المتوسط المرجح لقائمة أعداد ( انظر المشكلة رقم ٥٠ الجزء a من الفصل السادس ) .
- حساب الضرب التراكمى لقائمة أعداد ( انظر المشكلة رقم ٥٠ الجزء b من الفصل السادس ) .
- حساب المتوسط الهندسى لقائمة أعداد ( انظر المشكلة رقم ٥٠ الجزء c من الفصل السادس ) .
- حساب وجدولة قائمة بالأعداد الأولية ( انظر المشكلة رقم ٥٠ الجزء f من الفصل السادس ) .

(هـ) حساب جيب الزاوية مستخدما الطريقة المذكورة ( انظر المشكلة رقم ٥٠ الجزء أ من الفصل السادس ) .

(و) حساب أقساط إعادة دفع القرض ( انظر المشكلة رقم ٥٠ الجزء ز من الفصل السادس ) .

(ز) تحديد متوسط درجات الامتحان لكل طالب في الفصل كما هو مذكور في المشكلة رقم ٦ - ٥٠ الجزء k .

(٥٠) اكتب برنامجا كاملا بلغة البسكال لحل كل مشكلة من المشاكل المذكورة أدناه . استخدم إجراءات وذوال كلما كان ذلك ممكنا . ترجم ونفذ كل برنامج باستخدام البيانات المعطاه مع وصف المشكلة .

(١) افرض أنك تضع مبلغا معيناً من المال A في حساب توفير مع بداية كل سنة لعدد n من السنين . فإذا كان معدل الفائدة % i سنويا ، فإن المبلغ المتراكم بعد انقضاء سنة n هو F ، حيث :

$$F = A [(1+i/100) + (1+i/100)^2 + (1+i/100)^3 + \dots + (1+i/100)^n]$$

اكتب برنامج بسكال على هيئة حوار لتحديد مايلي :

(١) المبلغ المتراكم بعد 30 سنة إذا كان المبلغ المودع في بداية كل سنة هو 1000 دولار ، ومعدل الفائدة 6% في السنة .

(٢) المبلغ الذي يجب إيداعه في بداية كل سنة لكي يصبح المبلغ المتراكم 100,000 دولار بعد انقضاء 30 سنة بافتراض أن معدل الفائدة 6% سنويا .

حدد في كل حالة المبلغ غير المعروف أولا ، ثم اعمل جدولاً يوضح إجمالى المبلغ الذى يتراكم مع نهاية كل سنة . استخدم الدالة المكتوبة في المشكلة رقم ٧ - ١٤١ لأداء العمليات الأسية .

(ب) عدل البرنامج السابق لحساب الفائدة المركبة كل ربع سنة ، بدلا من حسابها كل سنة . قارن النتائج التى تحصل عليها في كل من الحالتين . ملاحظة : الصيغة المناسبة هي :

$$F = A [(1 + i/100m)^m + (1 + i/100m)^{2m} + (1 + i/100m)^{3m} + \dots + (1+i/100m)^{nm}]$$

حيث m تمثل عدد الفترات التى يدفع عنها فوائد في السنة .

(ج) تتحدد تكاليف قرض البناء بطريقة تجعل المقترض يدفع نفس المبلغ ( قسط ثابت ) للمؤسسة الدائنة شهريا طوال فترة سداد القرض . ويتغير جزء القسط الشهري المطلوب كفاية على الموازنة القائمة على أية حال من شهر لشهر . ففي بداية فترة سداد القرض يكون معظم القسط ممثلا لجزء الفائدة ، وجزء بسيط يمثل مايسدد من صلب القرض . وبالتدريج تصبح الموازنة القائمة أقل ، مما يقلل من الفائدة الشهرية ، وزيادة الجزء المسدد من صلب القرض . وعلى هذا ... تقل موازنة القرض بمعدل متزايد .

ويعرف الذين يشترون المنازل المبلغ الذين في حاجة لاقتراضه ، والوقت اللازم لإعادة دفعه . بعد ذلك ... فإنهم يطلبون القروض من مؤسسات الإقراض ، بعد معرفتهم القسط الشهري وقيمة الفائدة . كما يجب أن يهتموا أيضا بقيمة الفائدة المسددة شهريا ، وقيمة الجزء المسدد شهريا من صلب القرض ، ومايتبقى عليهم ، وذلك في كل شهر .

اكتب برنامجا بلغة البسكال يمكن أن تستخدمه مؤسسات الإقراض لتوفير هذه المعلومات للعملاء . افرض أن قيمة القرض ومعدل الفائدة السنوى وفترة سداد القرض محددة . القسط الشهري يحسب كما يلي :

$$A = \frac{iP(1+i)^n}{(1+i)^n - 1}$$

حيث :  $A$  = القسط الشهري بالدولار .

$P$  = إجمالي قيمة القرض بالدولار .

$i$  = معدل الفائدة الشهري ككسر عشري

( أى أن 1/2% تكتب 0.005 ) .

$n$  = إجمالي عدد الأقساط الشهرية .

ويمكن حساب مايدفع كفائدة شهرية من العلاقة التالية :

$$I = iB$$

حيث  $I$  = مايدفع كفائدة شهرية بالدولار .

$B$  = الموزنة القائمة الحالية بالدولار .

الموازنة القائمة الحالية تساوى المبلغ الاصلى للقرض ، مطروحا منه مجموع المدفوعات السابقة ناحية الأساس . مايدفع شهريا ناحية الأساس ( أى المبلغ المستخدم لتقليل الموزنة القائمة ، أو لتقليل صلب القرض ) هو :

$$T = A - I$$

حيث  $T$  = مايدفع شهريا ناحية الأساس .

استخدم الدالة المكتوبة فى المشكلة رقم ٧ - ٤١ لأداء العمليات الآسية .

استخدم البرنامج فى حساب تكلفة قرض 50,000 دولار ، يسدد على 25 سنة ، بمعدل فائدة سنوية 8% . ثم أعد الحسابات بمعدل فائدة 8.5% . ماهى درجة معنوية 0.5% فى معدل الفائدة عبر فترة سداد القرض ؟

(د) الطريقة المستخدمة فى حساب تكلفة قرض المنزل فى المشكلة السابقة تعرف بأنها طريقة القسط الثابت - con-stant payment method ، حيث إن كل الأقساط الشهرية متساوية . افرض أنه بدلا من ذلك يحتسب القسط الشهري باستخدام طريقة الفائدة البسيطة . أى افرض أن نفس الكمية تطبق ناحية تقليل مقدار القرض كل شهر ، أى

$$T = P/n$$

على أية حال تعتمد الفائدة الشهرية على قيمة الموزنة القائمة ، أى أن

$$I = iB$$

وعلى هذا ... فإن إجمالي ما يدفع شهريا A هو

$$A = T + I$$

سوف يقل في كل شهر حتى ينتهي القرض .

اكتب برنامجا بلغة البسكال لحساب تكلفة قرض مستخدما هذه الطريقة في إعادة الدفع . اكتب عناوين واضحة للمخرجات . استخدم البرنامج في حساب تكلفة قرض قيمته 50,000 دولار ، وفترة سداد 25 سنة ، ومعدل الفائدة 8 % سنويا . قارن النتائج بالنتائج التي تحصل عليها من المشكلة رقم ٧ - ٥ (ج) .

(هـ) افرض أننا لدينا عدد من النقاط اللثابة discrete  $(x_1, y_1)$   $(x_2, y_2)$  ... و  $(x_n, y_n)$  ، والتي نقرأ من منحنى  $y = f(x)$  ، حيث x محاطة بين  $x_1, x_n$  ، ونريد أن نقرب المساحة تحت المنحنى ، وذلك بتجزئة المنحنى إلى عدد من المستطيلات الصغيرة ، وحساب مساحات هذه المستطيلات ( هذا ما يعرف بقاعدة شبه المنحرف trapezoidal rule ) . والصيغة المناسبة هي :

$$A = (y_1 + y_2)(x_2 - x_1)/2 + (y_2 + y_3)(x_3 - x_2)/2 + \dots + (y_{n-1} + y_n)(x_n - x_{n-1})/2$$

لاحظ أن متوسط ارتفاع كل مستطيل يعطى بالعلاقة

$$(y_i + y_{i+1})/2$$

وأن عرض المستطيل يساوى

$$(x_{i+1} - x_i); i = 1, 2, \dots, (n-1)$$

حيث :

اكتب برنامجا بلغة البسكال لتنفيذ هذه العملية ، مستخدما دالة في تقويم الصيغة الرياضية  $y = f(x)$  ، استخدم البرنامج في حساب المساحة تحت المنحنى  $y = x^3$  بين الحدود  $x = 1, x = 4$  . حل هذه المشكلة أولا مستخدما 16 نقطة على أبعاد متساوية من بعضها ، ثم مستخدما 61 نقطة . وبعد ذلك مستخدما 301 نقطة . لاحظ أن دقة الحل تتحسن مع زيادة عدد النقاط . ( الإجابة الدقيقة لهذه المشكلة هي 63.75 ) .

(و) تصف المشكلة السابقة قاعدة شبه المنحرف trapezoidal rule لحساب المساحة تحت المنحنى  $y(x)$  ، حيث تستخدم مجموعة من القيم المجدولة  $(x_1, y_1)$   $(x_2, y_2)$  ...  $(x_n, y_n)$  في وصف المنحنى . إذا كانت القيم المجدولة لـ x على مسافات متساوية بعضها البعض ، فإن المعادلة المعطاه في المشكلة السابقة يمكن تبسيطها لتأخذ الشكل التالي :

$$A = (y_1 + 2y_2 + 2y_3 + 2y_4 + \dots + 2y_{n-1} + y_n)h/2$$

حيث h هي المسافة بين قيمتين متتاليتين من قيم x .

وهناك طريقة تستخدم عندما يكون هناك عدد زوجي من المناطق المتساوية ، أى عدد فردى من نقاط البيانات ، وهي قاعدة سيمبسون Simpson's rule . ومعادلة الحسابات المستخدمة مع قاعدة سيمبسون هي :

$$A = (y_1 + 4y_2 + 2y_3 + 4y_4 + 2y_5 + \dots + 4y_{n-1} + y_n)h/3$$

إحدى قيم  $h$  المعرفة ، فإن هذه الطريقة تعطي نتيجة أكثر دقة من طريقة قاعدة شبه المنحرف ( لاحظ أن الطريقة تحتاج لنفس كمية الحسابات التي تحتاجها طريقة شبه المنحرف تقريباً ) .

اكتب برنامجاً بلغة البسكال لحساب المساحة تحت المنحنى ، مستخدماً أى طريقة من الطرق سالفة الذكر ، مفترضاً أن هناك عدداً فردياً لنقاط بيانات متساوية المسافة بينها . نفذ كل طريقة بإجراء منفصل ، مستخدماً دالة  $y(x)$  في تقويم  $y$  .

استخدم البرنامج في حساب المساحة تحت المنحنى الممثل بالمعادلة .

$$y = e^{-x^2}$$

حيث  $x$  تتراوح من 0 إلى 1 . احسب المساحة مستخدماً كل طريقة ، ثم قارن النتائج بالنتيجة الصحيحة ، وهى :

$$A = 0.7468241$$

(ز) مازال هناك طريقة أخرى لحساب المساحة تحت المنحنى تستخدم أسلوب مونت كارلو Monte Carlo ، والتي تستخدم إنتاج أعداد عشوائية . افترض أن المنحنى  $y = f(x)$  موجب لأى قيمة من قيم  $x$  التي تقع بين حد أدنى محدد  $x = a$  ، وحد أعلى محدد  $x = b$  . دع أكبر قيمة لـ  $y$  داخل هذه الحدود هى  $y^*$  . وتنفذ طريقة مونت كارلو كما يلي :

(١) ابدأ بعدد قيمته تساوى صفرأ .

(٢) انتج عدداً عشوائياً  $r_x$  ، تقع قيمته بين  $a$  ،  $b$  .

(٣) قوم  $y(r_x)$

(٤) انتج عدداً عشوائياً  $r_y$  ، تقع قيمته بين  $0$  ،  $y^*$  .

(٥) قارن  $r_x$  ،  $r_y$  . فإذا كان  $r_y$  أقل من أو يساوى  $r_x$  : فإن هذه النقطة تقع على المنحنى أو تحته . وعند ذلك تزداد قيمة العداد بمقدار 1 .

(٦) أعد الخطوات من 2 إلى 5 عدداً كبيراً من المرات . وكل مرة تسمى دورة cycle .

(٧) وعند الانتهاء من إتمام عدد معين من الدورات ، فإن كسر النقاط الذى يقع على المنحنى أو تحته . والمسمى  $F$  يحسب بأنه قيمة العداد مقسوماً على إجمالى عدد الدورات . ويتم الحصول على ذلك على المساحة تحت المنحنى كمايلي :

$$A = Fy^*(b - a)$$

اكتب برنامجا بلغة البسكال لتنفيذ هذه العمليات . استخدم هذا البرنامج فى إيجاد المساحة تحت المنحنى  $y = e^{-x-2}$  التى تقع بين الحدين  $a=0$  ,  $b=1$  . حدد عدد الدورات اللازمة للحصول على إجابة دقيقة لأقرب ثلاثة أرقام معنوية . قارن وقت الكمبيوتر اللازم لهذه المشكلة مع الوقت اللازم للمشكلة السابقة . أى طريقة أفضل ؟

(ح) يمكن إنتاج التوزيع الطبيعي للتغير العشوائى  $x$  والذى له متوسط  $\mu$  وانحراف معيارى  $\sigma$  ، وذلك باستخدام الصيغة التالية :

$$x = \mu + \sigma \frac{\sum_{i=1}^N r_i - N/2}{\sqrt{N/12}}$$

حيث  $r_i$  هو عدد عشوائى له توزيع طبيعى ، وتقع قيمته بين 0 , 1 . وغالبا مايتكرر اختيار قيمة 12 عند استخدام هذه الصيغة . وأساس هذه الصيغة هو نظرية الحد المركزى central limit theorem ، والتى تحدد قيما للمتوسط لتغيرات عشوائية ذات توزيع طبيعى .

اكتب برنامجا بلغة البسكال ينتج عدداً محدداً من التغيرات العشوائية الموزعة توزيعاً طبيعياً ، لها متوسط معروف ، وانحراف معيارى معروف . اجعل عدد التغيرات العشوائية والمتوسط والانحراف المعيارى مؤشرات مدخلات .

(ط) اكتب برنامجا بلغة البسكال يسمح لشخص أن يلعب لعبة tic - tac - toe مع الكمبيوتر . اكتب البرنامج بحيث يمكن للكمبيوتر أن يكون اللاعب الأول أو اللاعب الثانى . فإذا ماكان الكمبيوتر هو اللاعب الأول ، دع أول حركة يتم إنتاجها عشوائياً . اكتب الحالة الكاملة للمباراة بعد كل حركة . دع الكمبيوتر يحدد من الذى يكسب عند حدوث ذلك .

(٥١) اكتب برنامجا كاملا بلغة البسكال لكل مشكلة من المشاكل التالية ، على أن يحتوى على دالة إعادة ذاتية .

(i) تحديد قيمة عدد فابوناكى  $F_n$  رقم  $n$  ، حيث :

$$F_n = F_{n-1} + F_{n-2} \text{ and } F_1 = F_2 = 1$$

( انظر المشكلة رقم ٥٠ - ٦ الجزء هـ ) . دع قيمة  $n$  كمؤشر مدخلات ، وهى أكبر من 2 .

(ب) يمكن حساب Legendre polynomials بواسطة الصيغ  $P_0 = 1, P_1 = x$  ،

$$P_n = \frac{2n-1}{n} x P_{n-1} - \frac{n-1}{n} P_{n-2}$$

حيث  $n = 2, 3, 4, \dots$  ،  $x$  أى عدد حقيقى يقع بين -1 , 1 ( لاحظ أن Legendre polynomials هي كميات من النوع الحقيقى ) . دع قيم  $n, x$  مؤشرات مدخلات .



## الفصل الثامن

### البيانات البسيطة التي يعرفها المستخدم

## User-Defined Simple-Type Data

سبق أن رأينا بالفعل أن هناك نوعين من البيانات البسيطة العامة في البسكال ، وهما : بيانات قياسية Standard - type data وبيانات يعرفها المستخدم user - defined data ( انظر القسم رقم ٢ من الفصل السابع ) . ويمكن تجزئة هذين النوعين إلى مايلي :

١ - بيانات قياسية :

أ - صحيحة

ب - حقيقية

ج - حرفية .

د - بولياني .

٢ - بيانات يعرفها المستخدم :

أ - نوع متعدد enumerated

ب - نوع مدى جزئي subrange

وقد سبق مناقشة استخدام البيانات القياسية بالتفصيل في الفصل الثالث من الكتاب . ونركز انتباهنا الآن إلى البيانات البسيطة التي يعرفها المستخدم . وسوف نعتبر بصفة خاصة البيانات المتعددة ، والبيانات من نوع المدى الجزئي . وسوف يتضح لنا أنه يمكن توضيح البيانات البسيطة ، وكيفية استخدامها بكفاءة في برنامج البسكال .

### ١ - بيانات من النوع المتعدد : 1. ENUMERATED - TYPE DATA

تحتوي البيانات المتعددة على ترتيب تسلسل من المعارف ، حيث يفسر كل معرف كعنصر بيانات مستقل يصاحب عناصر البيانات هذه مع بعضها اسم خاص يقوم بتعريف نوع البيانات . ومصاحبة اسم نوع البيانات مع عناصر البيانات الفردية يتحدد في تعريف النوع type . ويكتب تعريف النوع بصفة عامة كمايلي :

$TYPE\ name = (data\ item\ 1, data\ item\ 2, \dots, data\ item\ n)$

حيث name هو اسم بيانات من النوع المتعدد ، و data item 1 و data item 2 هي أسماء البيانات الفعلية .

مثال (٨-١)

نعرف في هذا المثال عنصر البيانات من النوع المتعدد ، اسمه days . وعناصر البيانات الفعلية تصبح أيام

الأسبوع ، وهي :

sunday, monday, tuesday, wednesday, thursday,  
friday and saturday

وعلى هذا ... فتعريف النوع يكتب كما يلي :

TYPE days = (sunday, monday, tuesday, wednesday, thursday, friday, saturday);

لاحظ أن عناصر البيانات مكتوبة بين قوسين ومفصولة عن بعضها بفواصل .

ويجب أن يكون مفهوما أن هذه هي القيم الوحيدة التي يمكنها أن تصاحب نوع days .

حيث إن البيانات من النوع المتعدد لها تعريف بترتيب متسلسل ، فيمكن استخدام المؤثرات العلاقية معها في تكوين تعبيرات بوليان . كما يمكن استخدام الدالتين القياسيتين succ , pred لتحديد أى عناصر البيانات تسبق عنصر بيانات محدد ، وأيها يتبعه .

مثال (٨-٢)

اعتبر days ، وهو من بيانات النوع المتعدد الذي سبق تعريفه في المثال السابق . وفيما يلي العديد من تعبيرات بوليان التي تستخدم عناصر البيانات وقيمها المناظرة .

التعبير	القيمة
sunday < tuesday	صحيح
wednesday >= saturday	خطأ
monday <> friday	صحيح
pred(friday) = thursday	صحيح
succ(friday) = saturday	صحيح
succ(tuesday) <> pred(thursday)	خطأ

ويمكن استخدام الدالة القياسية ord أيضا مع البيانات من النوع المتعدد . ولعمل ذلك يجب أن يكون مفهوما أن أول عنصر بيانات يحدد له الرقم 0 ، والثاني يحدد له الرقم 1 وهكذا . وعلى هذا ... فإذا كان هناك n عنصر بيانات داخل بيانات من النوع المتعدد ، فإن رقم الترتيب يتراوح من 0 إلى n-1 .

مثال (٨-٣)

اعتبر مرة أخرى days كبيانات النوع المتعدد ، والمعرفة في مثال (٨-١) . العلاقات التالية صحيحة :

succ(sunday) = monday	ord(sunday) = 0
pred(monday) = sunday	ord(monday) = 1
succ(monday) = tuesday	ord(tuesday) = 2

```

pred(friday) = thursday          ord(saturday) = 6
succ(friday) = saturday
pred(saturday) = friday

```

## ٢ - بيانات من نوع المدى الجزئي : 2. SUBRANGE - TYPE DATA

يشير المدى الجزئي subrange إلى جزء من مدى أصلي لنوع بيانات بسيط ومرتب . وبيانات المدى الجزئي هي عناصر بيانات تقع داخل هذا المدى الجزئي ، مكونة فئة جزئية من البيانات المرتبة والمتماسكة contiguous . ويشار إلى نوع البيانات الأصلي بأنه بيانات من النوع المضيق host . وكل عنصر بيانات من العناصر التي تقع داخل المدى الجزئي يعتبر من النوع المضيق .

ويمكن تطبيق مفهوم المدى الجزئي على أي مجموعة بيانات بسيطة مرتبة . ويشمل ذلك البيانات من النوع المتعدد التي سبق تعريفها ، كما يشمل ثلاثة أنواع من البيانات القياسية ، وهي البيانات الصحيحة والحرفية والبوليان . ( لاحظ أن البيانات الحقيقية لا يمكن cannot استخدامها في تعريف مدى جزئي ) .

والصيغة العامة لتعريف المدى الجزئي تكتب كما يلي :

```
TYPE name = first data item..last data item
```

حيث name هو اسم نوع بيانات المدى الجزئي . كما أن first data item هو أول عنصر بيانات في الترتيب داخل المدى الجزئي ، و last data item هو آخر عنصر بيانات في الترتيب داخل المدى الجزئي . ويحتوي المدى الجزئي الكامل على كل عناصر البيانات الموجودة بين هذين الحدين ( بما فيها الحدين أيضا ) . لاحظ أنه يجب أن يفصل أول عنصر بيانات عن آخر عنصر بيانات بوجود نقطتين متتاليتين .

### مثال (٤-٨)

فيما يلي عدة بيانات من النوع الذي يعرفه المستفيد .

```

TYPE days = (sunday,monday,tuesday,wednesday,thursday,friday,saturday);
weekdays = monday..friday;
month = 1..31;
caps = 'A'..'Z';

```

يعيد أول سطر تعريف النوع المتعدد المعطى في مثال (٨ - ١) . ونعرف في السطر الثاني نوع بيانات من المدى الجزئي يسمى weekdays يحتوي على فئة جزئية من الأيام (وهي بصفة خاصة عناصر البيانات الخمسة friday , thursday , wednesday , tuesday , monday) . لاحظ أن تعريف النوع المتعدد days يجب أن يسبق تعريف نوع المدى الجزئي weekdays .

ويعرف السطر الثالث نوع مدى جزئي آخر يسمى month . ويحتوي هذا النوع من البيانات على أرقام متتالية من ١ إلى ٣١ . وعلى هذا ... فإن نوع البيانات يكون فئة جزئية من النوع القياسي للبيانات الصحيحة .

أخيرا يعرف السطر الأخير نوع البيانات من المدى الجزئى ، وهو Caps الذى يحتوى على الحروف العلوية (من A إلى Z) . لاحظ أن آخر نوع من البيانات هذا هو فئة جزئية من البيانات الحرفية القياسية .

وحيث إن البيانات من نوع المدى الجزئى دائما ما تكون مرتبة ، فيمكن استخدام عناصر البيانات الفردية مع المؤثرات العلاقية لتكوين تعبيرات بوليان . كما يمكن استخدامها أيضا مع الدوال القياسية ord , succ , pred بنفس الطريقة التى تستخدم بها هذه الدوال مع البيانات البسيطة المرتبة الأخرى . وسوف نرى بعض أمثلة توضح استخدام بيانات من نوع المدى الجزئى فيما بعد فى هذا الفصل .

### 3. MORE ABOUT DECLARATIONS

#### ٣ - المزيد عن التوضيحات :

دعنا نستعرض مرة أخرى الهيكل الشامل لبرنامج البسكال . دعنا نفحص بصفة خاصة جزء التوضيحات من البرنامج ، حيث إننا قادرون الآن على فهم معنى كل العناصر التى يحتويها . ( تذكر أن هذه المادة سبق تقديمها فى القسم الخامس من الفصل الأول وفى القسم الثانى من الفصل الخامس .

فيما يلى تخطيطا شاملا للبرنامج :

١ - عنوان .

٢ - صلب البرنامج

أ - توضيحات :

١ - عناوين .

٢ - ثوابت

٣ - تعريفات الأنواع .

٤ - متغيرات .

٥ - إجراءات ودوال .

ب - عبارات .

ويبدأ قسم التوضيحات بعناوين العبارات ، والتى تستخدم مع عبارات GOTO ( انظر القسم الثامن من الفصل السادس ) . ويتبع ذلك تعريفات الثوابت ( القسم الثامن من الفصل الثانى ) ، ثم تعريفات الأنواع ( القسمين الأول والثانى من الفصل الثامن ) . ثم يلى ذلك توضيحات المتغيرات ( القسم التاسع من الفصل الثانى ) ، يتبعها توضيحات الإجراءات والدوال ( القسمين الأول والرابع من الفصل السابع ) .

وحقيقية أن تعريفات النوع تسبق توضيحات المتغيرات لها معنى خاص . فهذا يسمح بتوضيح المتغيرات بالنسبة لأنواع البيانات التى يعرفها المستفيد ، وبالنسبة أيضا لأنواع البيانات القياسية . ومقدرتنا على تمييز أنواع البيانات التى يعرفها المستفيد لها على ذلك أهمية كبيرة .

ومن الممكن أيضا دمج تعريف نوع المدى الجزئى المستخدم مع توضيح المتغير المناظر . وهذا مريح بصفة خاصة إذا ما استخدم نوع المدى الجزئى توضيحا واحدا فقط لمتغير .

وتوضيح المتغيرات التى تكون بياناتها من النوع الذى يحدده المستفيد موضح فى المثال التالى .

## مثال (٨-٥)

اعتبر تعريفات النوع وتوضيحات المتغيرات التالية : ( أعد إنتاج تعريفات النوع من مثال ٨ - ٤ ) .

```
TYPE days = (sunday,monday,tuesday,wednesday,thursday,friday,saturday);
  weekdays = monday..friday;
  month = 1..31;
  caps = 'A'..'Z';
VAR workdays,holidays : weekdays;
  dayofmonth : month;
  hoursworked : 1..24;
  grosspay,netpay : real;
  employeenumber : 1..maxint;
```

لاحظ أن المتغيرات weekdays , holidays هي من نوع weekdays . وعلى هذا ... فكل من هذه المتغيرات يمكن أن يأخذ قيمة monday , tuesday , wednesday , thursday , friday . وبالمثل المتغير dayofmonth هو من نوع month . وعلى هذا ... يمكن أن يأخذ أى قيمة صحيحة تتراوح من 1 إلى 31 .

أعتبر الآن المتغير hoursworked . هذا متغير من نوع المدى الجزئى ، ويشبه dayofmonth الذى يمكن أن يأخذ أى قيمة صحيحة تتراوح من 1 إلى 24 . لاحظ على أية حال أن تعريف النوع وتوضيح المتغير مندمجان معا فى هذه الحالة . وقد كان من الممكن بالطبع استبدال هذا التوضيح الفردى بما يلى :

```
TYPE hour = 1..24;
VAR hoursworked : hour;
```

بالرغم من أن هذا متعب بعض الشيء .

لاحظ أنه بإمكاننا أيضا أن ندمج تعريف نوع month مع توضيح المتغير dayofmonth . وعلى هذا ... يمكننا كتابة ما يلى :

```
VAR dayofmonth : 1..31;
```

وأخيرا فإن آخر توضيحين لمتغيرات يحددان أن grosspay , netpay متغيرات من النوع الحقيقى ، وأن employeenumber متغير موجب من النوع الصحيح . ( لاحظ أن employeenumber متغير من نوع المدى الجزئى فعلا ) وعلى هذا نرى توضيحات متغيرات تحتوى على أنواع بيانات قياسية ، وعلى أنواع بيانات يعرفها المستخدم أيضا فى هذا المثال .

#### ٤ - استخدام البيانات التى يعرفها المستخدم : - UTILIZING USER DEFINED DATA

بعد تعريف البيانات من النوع المتعدد ونوع المدى الجزئى ، يمكن استخدام هذه البيانات ، مثل عناصر البيانات بسيطة النوع الأخرى داخل برنامج البسكال . وهذا يشمل استخدامها كمتمغيرات تحكم control variables فى مكونات FOR وقائمين بالاختيار فى مكونات CASE .

### مثال (٨-٦)

فيما يلي مثالا لبيانات من النوع المتعدد مستخدمة كمتغيرات تحكم في مكون FOR .

```
FOR weekdays := monday TO friday DO
BEGIN
.
.
.
END;
```

متغير التحكم weekdays معرف في مثال (٨-٥) . وعلى هذا ... فإن weekdays يأخذ قيمة monday أثناء أول مسار في الدورة ، ثم يأخذ tuesday أثناء المسار الثاني للدورة وهكذا ، حتى يأخذ friday أثناء آخر ( خامس ) مسار .

### مثال (٨-٧)

اعتبر الآن استخدام نفس المتغير المتعدد weekdays كقوائم بالاختيار في مكون CASE .

```
CASE weekdays OF
monday : writeln(' First workday');
tuesday : writeln(' Second workday');
wednesday : writeln(' Third workday');
thursday : writeln(' Fourth workday');
friday : writeln(' Last workday')
END;
```

تكتب رسالة واحدة فقط في كل مرة يقابل فيها مكون CASE طبعا للقيمة التي تكون محدودة للمتغير weekdays .

سبق أن رأينا أنه يمكن استخدام بيانات متعددة النوع ، ومن نوع المدى الجزئي في إنتاج تعبيرات بولياني .

يمكن أيضا لبيانات المدى الجزئي من النوع الصحيح أن تظهر في تعبيرات من النوع الصحيح ، وتتبع بالضرورة نفس القواعد التي تتبعها البيانات القياسية صحيحة النوع .

ويمكن أن تحدد للمتغيرات من النوع المتعدد ، ومن نوع المدى الجزئي قيم مناظرة لها من النوع المناسب . وعلى هذا ... يمكن تحديد أحد عناصر النوع المتعدد لمتغير متعدد من نفس النوع . وبالمثل يمكن تحديد عنصر من عناصر المدى الجزئي لمتغير مدى جزئي من نفس النوع .

### مثال (٨-٨) :

افرض أن holidays متغير متعدد ، وأن employeenumber , hoursworked متغيران مدى جزئي كما هي معرفة في مثال (٨-٥) . وعلى هذا ... فإن عبارات التحديد التالية صحيحة :

```
employeenumber := 12345;
holidays := thursday;
hoursworked := 8;
```

لا يمكن إدخال البيانات متعددة النوع داخل الكمبيوتر عن طريق عبارة read أو readln . وعلى أية حال ... يمكن إدخال البيانات من نوع المدى الجزئي صحيحة أو حرفية لتوفر قيم تقع داخل المدى الجزئي المسموح به . ونفس القيود موجودة على كتابة البيانات من الكمبيوتر عن طريق عبارة write أو writeln .

ويسمح استخدام البيانات من النوع المتعدد أن يعبر عن برنامج اليسكال بالنسبة لمتغيرات مرتبطة ارتباطا وثيقا بتعريف المشكلة المناظر . وهذه العملية يوصى باستخدامها ، حيث إنها تزيد من الوضوح ومن بساطة المنطق . كما أن ذلك يسهل من تصحيح أخطاء البرنامج .

ومزايا استخدام بيانات المدى الجزئي شبيهه بذلك أيضا ، بالرغم من قلة تأثيرها . وبصفة خاصة فإن استخدام بيانات المدى الجزئي تبسط البرنامج وتحسن من قراءته ، كما يمكن أن ينتج عنها تقليل في مساحة الذاكرة المطلوبة .

### مثال (٨-٩)

عدد الأيام بين تاريخين . افترض أنه تم إدخال تاريخين مختلفين داخل الكمبيوتر ، وأنت تريد تحديد عدد الأيام الموجودة بين هذين التاريخين . إحدى طرق علاج هذه المشكلة هي تحديد عدد الأيام منذ كل من التاريخين ، وحتى تاريخ أساسي مشترك سابق لهما . ويمكن عند ذلك تحديد عدد الأيام الموجود بين هذين التاريخين بطرح الرقمين من بعضهما .

دعنا نختار أول يناير 1960 كتاريخ أساسي . إذا ما أدخل التاريخان في صورة yy dd mm ( حيث mm تمثل رقما صحيحا يعبر عن الشهر ، dd رقما آخر صحيحا يعبر عن اليوم ، yy رقما ثالثا صحيحا يعبر عن السنة ) ، فإن عدد الأيام منذ تاريخ الأساسي حتى التاريخ المعطى يمكن تحديده بالطريقة التالية :

١ - حدد اليوم من السنة الحالية بالتقريب كما يلي :

$$n := \text{trunc}(30.42 * (mm - 1)) + dd$$

٢ - إذا كان mm = 2 ( شهر فبراير ) أضف 1 إلى قيمة n .

٣ - إذا كان mm > 2 ، وكانت mm < 8 ( شهر مارس أو أبريل أو مايو أو يونيو أو يوليو ) انقص 1 من قيمة n

٤ - إذا كان yy MOD 4 = 0 ، mm > 2 ( سنة كبيسة ) أضف 1 إلى قيمة n .

٥ - أضف 1461 إلى قيمة n لكل دورة من أربع سنوات بعد أول يناير 1960 .

٦ - أضف 365 إلى قيمة n لكل سنة إضافية بعد إتمام آخر دورة من أربع سنوات ، ثم أضف 1 ( لأحدث سنة كبيسة ) .

دعنا نستخدم دالة اسمها daysbeyond 1960 لتحديد قيمة مناسبة لـ n ، وذلك بمعرفة قيم yy dd mm . نفرض أن n ، dd ، yy متغيرات مدى جزئي صحيحة . وفيما يلي دالة كاملة مبنية على الست خطوات سالفة الذكر .

```
FUNCTION daysbeyond1960(mm : month; dd : day; yy : year) : numberofdays;
(* This function converts a given date (month,day,year)
   to the number of days beyond January 1, 1960. *)
VAR n : numberofdays;
```

```
BEGIN
  n := trunc(30.42*(mm-1)) + dd;
  IF mm=2 THEN n := n + 1;
  IF (mm > 2) AND (mm < 8) THEN n := n - 1;
  IF (yy MOD 4 = 0) AND (mm > 2) THEN n := n + 1;
  IF (yy-1960) DIV 4 > 0 THEN n := n + 1461*((yy-1960) DIV 4);
  IF (yy-1960) MOD 4 > 0 THEN n := n + 365*((yy-1960) MOD 4) + 1;
  daysbeyond1960 := n
END;
```

ومن السهل الآن إكمال بقية محتويات البرنامج . لعمل ذلك ... فإننا نعرف أنواع المدى الجزئي الصحيحة ( days2 , year , month , day , numberofdays , ثم نوضح مجموعة من المتغيرات الشاملة من نوع المدى الجزئي , days1 , yy , mm , dd ) . ويلى ذلك توضيح الدالة ، ثم يليه المجموعة الرئيسية التى تقرأ التاريخين وتتصل بالدالة لكل تاريخ منهما ، ثم تكتب النتيجة النهائية

وفيما يلى محتويات البرنامج .

```
PROGRAM dates(input,output);

(* THIS PROGRAM DETERMINES THE NUMBER OF DAYS BETWEEN TWO DATES *)

TYPE day = 1..31;
    month = 1..12;
    year = 1960..2100;
    numberofdays = 0..maxint;
VAR dd : day;
    mm : month;
    yy : year;
    days1,days2 : numberofdays;

FUNCTION daysbeyond1960(mm : month; dd : day; yy : year) : numberofdays;
(* This function converts a given date (month,day,year)
   to the number of days beyond January 1, 1960. *)
VAR n : numberofdays;
BEGIN
  n := trunc(30.42*(mm-1)) + dd;
  IF mm=2 THEN n := n + 1;
  IF (mm > 2) AND (mm < 8) THEN n := n - 1;
  IF (yy MOD 4 = 0) AND (mm > 2) THEN n := n + 1;
  IF (yy-1960) DIV 4 > 0 THEN n := n + 1461*((yy-1960) DIV 4);
  IF (yy-1960) MOD 4 > 0 THEN n := n + 365*((yy-1960) MOD 4) + 1;
  daysbeyond1960 := n
END;

BEGIN (* main action block *)
  write(' Enter the first date (mm dd yyyy) :');
  readln(mm,dd,yy);
  days1 := daysbeyond1960(mm,dd,yy);
  write(' Enter the second date (mm dd yyyy) :');
  readln(mm,dd,yy);
  days2 := daysbeyond1960(mm,dd,yy);
  writeln;
  writeln(' There are ',days2-days1-1,' days between the two dates.')
END.
```



لاحظ أنه افترض أن التاريخ الثاني أكبر من التاريخ الأول . وينتج عن تنفيذ هذا البرنامج المخرجات التالية إذا ما كان تاريخا المدخلات هما 12 29 1963 و 3 21 1983 ( استجابة المستخدم موضوع تحت خط ) .

Enter the first date (mm dd yyyy) : 12 29 1963

Enter the second date (mm dd yyyy) : 3 21 1983

There are 7021 days between the two dates.

ويوضح المثال السابق استخدام بيانات المدى الجزئي في مشكلة برمجة فعلية . أما استخدام البيانات من النوع المتعدد ، فموضح في المثال التالي .

### مثال (٨-١٠)

برج هانوي . اعتبر مرة أخرى المشكلة المذكورة في مثال (٢٦) في الفصل السابع . دعنا نعدل الآن برنامج البسكال ، بحيث يستخدم بيانات من النوع المتعدد في تمثيل الثلاثة أعمدة . وبصفة خاصة دعنا نقدم بيانات من النوع المتعدد poles ، والتي تتكون من ثلاثة عناصر ، هي : right , center , left . وفيما يلي البرنامج كاملاً لهذه المشكلة :

```
PROGRAM towersofhanoi(input,output);

(* THIS PROGRAM SOLVES A WELL-KNOWN GAME USING
   RECURSIVE PROCEDURE CALLS AND USER-DEFINED DATA *)

TYPE poles = (left,center,right);
   disks = 0..maxint;
VAR n : disks;

PROCEDURE transfer(n : disks; origin,destination,other : poles);
(* Transfer n disks from the origin to the destination *)

PROCEDURE diskmove(origin,destination : poles);
(* Move a single disk from the origin to the destination *)
BEGIN
  write(' Move ');
  CASE origin OF
    left : IF destination = center
            THEN writeln('left to center')
            ELSE writeln('left to right');
    center : IF destination = left
              THEN writeln('center to left')
              ELSE writeln('center to right');
    right : IF destination = center
             THEN writeln('right to center')
             ELSE writeln('right to left')
  END (* case *)
END; (* diskmove *)

BEGIN (* transfer *)
  IF n>0 THEN BEGIN
    transfer(n-1,origin,other,destination);
    diskmove(origin,destination);
    transfer(n-1,other,destination,origin)
  END
END; (* transfer *)

BEGIN (* main action block *)
  write(' Enter the number of disks -> ');
  readln(n);
  writeln;
  transfer(n,left,right,center)
END.
```

. وهذه الصيغة للبرنامج ليست بنظافة الصيغة المقدمة في مثال ٧ - ٢٦ بسبب القيد الموضوع على البيانات من النوع المتعدد بأنها لا يمكن أن تظهر في عبارة write أو عبارة writeln . لاحظ بصيغة خاصة الحاجة إلى مكون CASE المرفق بعض الشيء في الإجراء diskmove ، وإلا فلن تعريف البرنامج على أية حال يكون مباشرا ، حيث إن الأعمدة الفردية يمكن أن يشار إليها بمصطلحات طبيعية أكثر .

ويعطى تنفيذ البرنامج في حالة أن  $n = 3$  المخرجات التالية . ( استجابات المستخدم موضوع تحتها خط ) .

Enter the number of disks -> 3

Move left to right  
Move left to center  
Move right to center  
Move left to right  
Move center to left  
Move center to right  
Move left to right

## Review Questions

## أسئلة للمراجعة :

- ١ - ماهما الفئتان العامتان للبيانات من النوع البسيط في البسكال ؟ وكيف يمكن تقسيم كل منهما ؟
- ٢ - كيف تعرف بيانات النوع المتعدد ؟
- ٣ - في أى مفهوم تحتوى بيانات النوع المتعدد على ترتيب متسلسل ؟
- ٤ - أى المؤثرات يمكن استخدامها مع بيانات النوع المتعدد ؟
- ٥ - أى الدوال القياسية يمكن استخدامها مع بيانات النوع المتعدد ؟
- ٦ - كيف تعرف بيانات المدى الجزئى ؟ قارنها مع بيانات النوع المتعدد .
- ٧ - إلى أى نوع من أنواع البيانات البسيطة يمكن تطبيق مفهوم المدى الجزئى ؟ ما نوع البيانات البسيطة غير المتوافقة مع مفهوم المدى الجزئى ؟
- ٨ - أى المؤثرات يمكن استخدامها مع بيانات المدى الجزئى ؟
- ٩ - أى الدوال القياسية يمكن استخدامها مع بيانات المدى الجزئى ؟
- ١٠ - ماهى العناصر التى توجد فى جزء توضيحات برنامج البسكال ؟ وفى أى ترتيب يجب أن تظهر هذه العناصر ؟
- ١١ - تحت أى ظروف يكون من المفيد دمج تعريف نوع المدى الجزئى مع توضيح المتغير المناظر ؟ هل يمكن عمل ذلك مع بيانات النوع المتعدد أيضا ؟
- ١٢ - هل يمكن استخدام عنصر بيانات من النوع المتعدد كمتغير تحكم فى مكون FOR ؟ وهل يمكن استخدام عنصر بيانات من نوع المدى الجزئى لنفس الغرض ؟

١٣ - هل يمكن استخدام عنصر بيانات من النوع المتعدد كقائم بالاختيار في مكون CASE ؟ وهل يمكن استخدام عنصر بيانات من نوع المدى الجزئي لنفس الغرض ؟

١٤ - ماهى القيود الموجودة على استخدام عناصر بيانات يعرفها المستفيد في التعبير ؟

١٥ - هل يمكن لعنصر بيانات يعرفه المستفيد أن يظهر في عبارة تحديد ؟ ماهى القيود التي تطبق في هذه الحالة ؟

١٦ - ماهى القيود الموضوعة على استخدام عناصر بيانات يعرفها المستفيد في عبارات مدخلات ومخرجات ؟

١٧ - ماهى مزايا استخدام بيانات النوع المتعدد في برامج البسكال ؟ وماهى عيوب ذلك ؟ وضع إجابتك .

١٨ - ماهى مزايا استخدام بيانات المدى الجزئي في برامج البسكال ؟ وماهى عيوب ذلك ؟ وضع إجابتك .

## Solved Problems

## مسائل محلولة :

١٩ - اعتبر تعريف النوع التالى :

فيما يلى عدة تعبيرات بوليان تستخدم عناصر البيانات هذه . حدد قيمة كل تعبير .

التعبير	القيمة
re > fa	- خطأ
do <= ti	- صحيح
pred(mi) = re	- صحيح
mi = succ(fa)	- خطأ
ord(do) < 1	- صحيح
3 = ord(pred(fa)) + 1	- صحيح

٢٠ - يوضح التخطيط الهيكلى التالى تعريف واستخدام بيانات من النوع المتعدد ، أو من نوع المدى الجزئي .

```
(a) PROGRAM sample1(input,output);
TYPE flavors = (vanilla,chocolate,strawberry,cherry,coconut);
    sizes = (small,medium,large);
VAR cone,dish,sundae : flavors;
    pint,quart : vanilla..strawberry;
    conesize : sizes;
BEGIN
.
.
cone := cherry;
.
.
IF quart = chocolate THEN . . .
ELSE . . .;
```

```

CASE conesize OF
    small : . . .;
    medium : . . .;
    large : . . .
END;
.
.
END.

(b) PROGRAM sample2(input,output);
    TYPE primary = (yellow,cyan,magenta);
    VAR color : primary;

    PROCEDURE proc1(VAR hue : primary);
    BEGIN
        .
        .
        IF hue = cyan THEN . . .
            ELSE . . .;
        .
        .
    END;

    BEGIN (* main action statements *)
        .
        .
        FOR color = yellow TO magenta DO . . .
        .
        .
    END.

(c) PROGRAM sample3(input,output);
    TYPE digits = '0'..'9';
    VAR d1,d2 : digits;
    BEGIN
        .
        .
        REPEAT
            .
            .
            .
        UNTIL d1 = d2;
        .
        .
    END.

```

لاحظ أن المتغيرات d1 , d2 لا تمثل كميات عددية .

## Supplementary Problems

## مشاكل متكاملة :

٢١ - اعتبر تعريف النوع التالي :

```
TYPE cities = (Boston,Miami,Pittsburgh,Chicago,Denver,Phoenix,Seattle);
```

حدد قيمة كل تعبير من التعبيرات التالية :

- |                        |                          |
|------------------------|--------------------------|
| (a) pred(Pittsburgh)   | (e) succ(ord(Chicago))   |
| (b) succ(Denver)       | (f) Miami < Seattle      |
| (c) ord(Seattle)       | (g) Chicago < Pittsburgh |
| (d) ord(succ(Chicago)) |                          |

٢٢ - فيما يلي تخطيطات هيكلية لبرامج بسكال ، بعضها مكتوب بطريقة خاطئة . عرف الأخطاء .

```
(a) PROGRAM sample(input,output);
    TYPE days = (sun,mon,tues,wed,thurs,fri,sat);
    VAR day1 : mon..fri;
        day2 : sun..sat;
    BEGIN
        .
        .
        day1 := mon;
        WHILE day1 <> day2 DO
            BEGIN
                .
                .
                day1 := succ(day1)
            END;
        .
        .
    END.
```

```
(b) PROGRAM sample(input,output);
    VAR highlight,foreground,background : colors
        main : red..blue;
    CONST border = red;
    TYPE colors = (red,green,blue,white,black);
    BEGIN
        .
        .
        .
        foreground := 2;
        background := pred(foreground)
        .
        .
    END.
```

```
(c) PROGRAM sample(input,output);
    TYPE suits = (clubs,diamonds,hearts,spades);
        values = 1..12;
    VAR cardtype : suits;
        cardvalue : values;
        count,n : 1..maxint;
    BEGIN
        .
        .
        write(' How many cards? ');
        readln(n);
```

```

FOR count := 1 TO n DO
  BEGIN
    .
    .
    cardtype := . . . ;
    cardvalue := . . . ;
    writeln(count,cardtype,cardvalue)
  END
END.

```

```

(d) PROGRAM sample(input,output);
    TYPE suits = (clubs,diamonds,hearts,spades);
        values = 1..12;
    VAR cardtype : suits;
        cardvalue : values;
        count,n : 1..maxint;
    BEGIN
        .
        .
        write(' How many cards? ');
        readln(n);
        .
        .
        FOR count := 1 TO n DO
            IF cardvalue > 9 THEN writeln('Picture card');
        .
        .
    END.

```

```

(e) PROGRAM sample(input,output);
    TYPE wines = (chablis,sauterne,rose,burgundy,chianti);
    VAR dinner : wines;
        cost : real;

    FUNCTION special(cost : real) : wines;
    BEGIN
        .
        .
        .
        special := . . .
    END;

    BEGIN (* main action statements *)
        .
        .
        readln(cost);
        dinner := special(cost);
        .
        .
    END.

```

## Programming Problems

## مشاكل مبرمجة :

٢٣ - عدل البرنامج الموضح في مثال (٨ - ٩) بحيث إنه يقبل المعلومات التالية كمدخلات .

١ - تاريخ اليوم .

ب - اسم المستفيد ( افترض أن الاسم مكون من خمسة حروف ) .

ج - تاريخ ميلاد المستفيد .

ثم يحدد بعد ذلك عدد الأيام التي عاشها المستفيد .

( أدخل التواريخ بنفس الطريقة المذكورة في مثال (٨ - ٩) )

٢٤ - عدل البرنامج الموضح في مثال (٨ - ٩) ، بحيث إنه يقبل أى تاريخ بعد أول يناير 1960 كمدخلات ، ثم يطبع اليوم المناظر في الأسبوع . ملاحظة : ١ يناير 1960 كان يوم جمعة Friday

٢٥ - عدل البرنامج الموحد في مثال (٨ - ١٠) بحيث إنه يمكن لمجموعة الأقراص أن تكون في أى عمود ( ولا يلزم أن تكون في العمود الموجود على أقصى اليسار ) ويمكن أن يكون العمود النهائى أى عمود من الأعمدة ( وليس بالضرورة أن يكون العمود الموجود على أقصى اليمين ) . حدد عمود الأصل وعمود المقصد كمؤشرات مدخلات . بعد ذلك اطبع الحركات المطلوبة لنقل الأعمدة من الأصل المجدد إلى الغاية المحددة .

٢٦ - عدل كل من البرامج الآتية ، مستخدما بيانات من النوع المتعدد ، أو من نوع المدى الجزئى ، مع استخدام إجراءات ودوال أيضا كلما كان ذلك ممكنا .

١ - إعادة حساب متوسط قائمة أعداد ( انظر مثال (٦ - ٩) ) .

ب - حساب قائمة مضروبوات ( انظر مثال (٦ - ١٠) ) .

ج - حساب الاستهلاك ( انظر مثال (٧ - ١٢) ) .

د - محاكاة مباراة ( انظر مثال (٨ - ١٨) ) .

٢٧ - حل كل مشكلة من مشاكل البرمجة التالية ، مستخدما بيانات من النوع المتعدد ، وبيانات من نوع المدى الجزئى كلما كان ذلك ممكنا ، تأكد أيضا من استخدام الأجزاء في البرنامج ، وذلك باستخدام الإجراءات والدوال استخدما مناسباً .

١ - حساب أول  $n$  عدد من أعداد فيبوناكى كما سبق ذكر ذلك في المشككة رقم ٥ - هـ فى الفصل ٦ . ثم تحديد أى هذه الأعداد عددا أوليا ( انظر المشككة رقم ٥ - و الفصل ٦ ) .

ب - تحويل عدد صحيح موجب إلى عدد روماني كما هو مذكور في المشككة رقم (٩) ٥٠ الفصل ٦ .

ج - برمجة مباراة متداخلة للعبة tic - tac - toe كما هو مذكور في المشككة رقم ( i ) ٥٠ الفصل ٧ .





## الفصل التاسع

### المنظومات

## Arrays

سبق أن أشرنا إلى أن البسكال يقدم العديد من فئات مختلفة لأنواع البيانات : بيانات بسيطة وبيانات مرتبة structured وبيانات مشيرات pointers . وحتى الآن اقتصرت مناقشتنا - على أية حال - بالبيانات بسيطة النوع . وقد رأينا أن البيانات بسيطة النوع تحتوى على أربعة أنواع : بيانات قياسية ، وهي البيانات الصحيحة integer ، والحقيقية real ، والحرفية char ، والبوليان boolean ( انظر الفصل الثالث ) كما أنها تحتوى على نوعين من البيانات التي يعرفها المستفيد أيضا ، وهي البيانات المتعددة enumerated ، وبيانات المدى subrange ( انظر الفصل الثامن ) . . . وجميع البيانات البسيطة لها خاصية مشتركة ، وهي أن كل متغير يمثل عنصر بيانات واحد .

ونعيد الآن انتباهنا إلى البيانات مرتبة النوع . وفي واقع الأمر ... كل فصل من الفصول الأربعة القادمة يهتم بنوع مختلف من أنواع البيانات المرتبة . وتغطي هذه الفصول المنظومات arrays ، والسجلات records ، والملفات files والفئات sets على التوالي . وجميع هذه الأنواع من البيانات لها خاصية مشتركة ، وهي أنه يمكن لمعرف واحد أن يمثل عناصر بيانات متعددة . كما يمكن أيضا الاتصال بعنصر البيانات الفردى بمفرده ( بالرغم من حدوث ذلك بطريقة مختلفة لكل نوع من أنواع البيانات المرتبة ) . وعلى هذا ... فعناصر البيانات المرتبة يمكن أن تعامل مع بعضها ، أو كل عنصر بمفرده ، وذلك طبقا لمتطلبات كل تطبيق خاص .

### 1. ONE - DIMNTIONAL ARRAYS

١ - منظومات ذات بعد واحد :

يمكن التفكير في المنظومة ذات البعد الواحد على أنها قائمة list ( أو عمود ) بعناصر البيانات ، وكلها من نفس النوع . ويشار إليها جميعها باسم واحد . وكل عنصر فردى في المنظومة ( أى كل عنصر من عناصر البيانات ) يمكن أن يشار إليه بتحديد اسم المنظومة ، يليه فهرس index ( والذي يسمى دليل subscript ) محصورا بين قوسين مربعين . ويمكن لعناصر المنظومة أن تكون من أنواع البيانات ، طالما أنها جميعها من نفس النوع . وعلى هذا ... إذا كانت القائمة list منظومة ذات بعد واحد تحتوى على ٧١ عنصرا ، فإن العناصر الفردية للمنظومة تكون list [2] ، ... ، list [n] list [1] . . . وعلى هذا ... يمكن تمثيل محتويات المنظومة كما يلي :

$$\text{list} = \begin{bmatrix} \text{list [1]} \\ \text{list [2]} \\ . \\ . \\ . \\ \text{list [n]} \end{bmatrix}$$

### مثال (٩-١)

يحتوى برنامج بسكال على منظومة من النوع الحقيقى بها 100 عنصر . كل عنصر فردى له قيمة تساوى 0.01 مضروبة فى قيمة الفهرس الخاص به . وعلى هذا ... فإن :

```
list [1] = 0.01
list [2] = 0.02
list [3] = 0.03
.
.
etc.
```

العنصر i فى list يمكن أن يشار اليه بأنه list [i] ، حيث i متغير صحيح . وعلى هذا ... تحدد قيمة 5 للمتغير i إذا ما أشار i الى العنصر الخامس فى المنظومة ، وهو list [5] . وقيمة هذا العنصر هى 0.05 .

وأسهل طريق لتعريف منظومة هو وضعها فى توضيح متغير كما يلى :

```
VAR array name : ARRAY [index type] OF type
```

ويمكن لنوع الفهرس أن يكون عددا ترتيبيا من النوع البسيط ( أى رقماً صحيحاً أو حرفياً أو بوليان أو متعدد ) أو من نوع المدى الجزئى . أما المنظومة نفسها ، فيمكن أن تكون من أى نوع ، بما فى ذلك الأنواع المرتبة ( سوف يذكر المزيد بعد ذلك عن هذا ) . بالرغم من أن معظم التطبيقات تستخدم منظومات بسيطة النوع فقط . ( كنوع من الحاجة العملية . لاحظ أن المنظومة يمكنها أن تكون من النوع الحقيقى ، إلا أن فهرسها لا يمكن أن يكون كذلك ) .

### مثال (٩-٢)

افرض أن برنامج البسكال يحتوى على منظومة اسمها list مكونة من 100 عنصر . يمكن كتابة توضيح المنظومة كما يلى :

```
VAR list : ARRAY [1..100] OF real;
```

كما يمكن كتابة توضيحاً بديلاً كما يلى :

```
TYPE index = 1..100;
VAR list : ARRAY [index] OF real;
```

وعادة ماتكون الصيغة الأولى أفضل ، نظراً لبساطتها .

لاحظ أن عناصر المنظومة من النوع الحقيقى . أما الفهرس ، فهو من نوع المدى الجزئى الصحيح .

يمكن استخدام العناصر الفردية للمنظومة فى تعبيرات أو عبارات تحديد وعبارات read , write وخلافه ، حيث أنها متغيرات معتادة بسيطة النوع . ولعمل ذلك يجب أن يكتب عنصر المنظومة على أنه اسم المنظومة ، تتبعه قيمة مناسبة للفهرس ، محصورة بين قوسين مربعين . ويمكن التعبير عن قيمة الفهرس ككاتب أو متغير أو تعبير . وعلى أية حال ... يجب أن يكون الفهرس من النوع الصحيح ( وليس الخاطئ ) كما يجب أن يقع داخل المدى الصحيح ( وليس الخاطئ ) للمنظومة .

## مثال (٩-٣)

دعنا نعتبر مرة أخرى منظومة list ذات البعد الواحد ، ومن النوع الحقيقي المذكورة في المثال السابق . افترض أننا نريد الآن فحص قيمة كل عنصر من عناصر المنظومة ، وكتابة القيم التي تكون سالبة . يمكن تحقيق ذلك بمكون الدورة التالي :

```
FOR count := 1 TO 100 DO
  IF list [count] < 0
    THEN writeln(count;3, list [count]:4:1);
```

حظ أن قيمة الفهرس ( أى المتغير صحيح النوع count ) تكتب على نفس السطر مع عنصر المنظومة السالب . وعلى هذا ... إذا كان ثالث عنصر من عناصر المنظومة له القيمة -5.0 ، فينتج عن ذلك سطر المخرجات التالي :

3 -5.0

وفي هذا المثال من الضروري ألا تقع قيم count خارج المدى من 1 إلى 100 ، حيث إن فهرس المنظومة غير معرف خارج هذا المدى . ويمكننا - على أية حال - العمل مع مدى جزئى أصغر إذا ما أردنا ذلك . وعلى هذا ... كان يمكننا كتابة :

```
FOR count := 25 TO 50 DO
  IF list [count] < 0
    THEN writeln(count;3, list [count]:4:1);
```

ويجب أن يتبع استخدام عناصر المنظومة فى عبارات تحديد أو فى تعبيرات نفس القواعد السارية على المتغيرات بسيطة النوع . وعلى هذا ... إذا ما ظهر عنصر منظومة فردى فى أى ناحية من عبارة تحديد ، فيجب أن يكون نوعا متوافقا مع المتغير الآخر أو التعبير الموجود فى هذه العبارة . وكذلك إذا ما استخدم فى تعبير ، فيجب أن يكون عنصر المنظومة من نوع متوافق مع بقية عناصر البيانات التى تظهر فى هذا التعبير .

## مثال (٩-٤)

الانحراف عن المتوسط . افترض أننا نريد قراءة قائمة من n كمية حقيقية ، ونحسب متوسطها كما فى مثال ٦ - ٨ . بالإضافة إلى ذلك ... فإننا نريد حساب الانحراف أيضا لكل عدد من المتوسط باستخدام العلاقة :

$$\text{deviation} = x[i] - \text{average}$$

حيث  $x[i]$  يمثل كل عدد من الأعداد المعطاة ، ويمثل average المتوسط المحسوب .

لاحظ أننا نخزن كل عدد من الأعداد المعطاة فى منظومة حقيقية النوع وذات بعد واحد . وهذا جزء ضرورى من هذا البرنامج . والسبب فى ذلك - الذى يجب أن يكون مفهوما جيدا - هو ما يلى :

فى كل الأمثلة السابقة التى حسبنا فيها المتوسط لقائمة أعداد ، كان كل عدد يحل محل العدد السابق له فى القائمة . وعلى هذا ... فقد كانت الأعداد الفردية غير متاحة لأى حسابات أخرى بعد الانتهاء من حساب المتوسط ، إلا أن هذه القيم الفردية يجب أن تحفظ الآن لحساب الانحراف المناظر ، وذلك بعد تحديد المتوسط . وعلى هذا ... فإنها تخزن فى منظومة x لها بعد واحد .

دعنا نقيّد حجم x إلى 100 عنصر . نحتاج على أية حال إلى استخدام كل العناصر المائة . بدلا من ذلك سوف نحدد العدد الفعلى للعناصر ، وذلك بتحديد كمية صحيحة موجبة ( لا تزيد عن 100 ) للمتغير n .

وفيما يلي برنامجاً كاملاً بلغة البسكال لأداء ذلك .

```
PROGRAM deviations(input,output);

(* THIS PROGRAM READS IN A LIST OF n NUMBERS,
   CALCULATES THEIR AVERAGE, AND THEN COMPUTES
   THE DEVIATION OF EACH NUMBER ABOUT THE AVERAGE *)

VAR n,count : integer;
    sum,average,deviation : real;
    x : ARRAY [1..100] OF real;

BEGIN
  BEGIN (* Read in the numbers and calculate the average *)
    write(' How many numbers will be averaged? ');
    readln(n);
    writeln;
    sum := 0;
    FOR count := 1 TO n DO
      BEGIN
        write(' i= ',count:3,'      x= ');
        readln(x [count]);
        sum := sum + x [count]
      END;
    average := sum/n;
    writeln(' The average is ',average);
    writeln;
  END;  (* calculate average *)

  BEGIN (* Calculate the deviations about the average *)
    FOR count := 1 TO n DO
      BEGIN
        deviation := x [count] - average;
        write(' i= ',count:3,'      x= ',x [count]);
        writeln('      d= ',deviation)
      END
    END  (* calculate deviations *)
END.
```

لاحظ طريقة عمل أجزاء هذا البرنامج ، حيث لا توجد إجراءات أو دوال في البرنامج . وبدلاً من ذلك ... تحتوي المجموعة الرئيسية على عبارتين مركبتين ، كل منهما لها هدفها المميز . فكل عبارة مركبة معرفة مع تعليقاتها الخاصة بها . وهذه الطريقة مريحة بالنسبة للبرامج البسيطة القصيرة مثل هذا البرنامج .

افترض أن البرنامج نفذ الآن باستخدام القيم العددية الخمس التالية :

$x[1] = 3.0$ ,  $x[2] = -2.0$ ,  $x[3] = 12.0$ ,  $x[4] = 4.4$ ,  $x[5] = 3.5$

يظهر إدخال البيانات المتداخل كما يلي ( استجابات المستفيد موضوع تحتها خط ) .

How many numbers will be averaged? 5

i=	1	x=	<u>3.0</u>
i=	2	x=	<u>-2.0</u>
i=	3	x=	<u>12.0</u>
i=	4	x=	<u>4.4</u>
i=	5	x=	<u>3.5</u>

ويمجرد إدخال جميع البيانات ، سوف تظهر النتائج التالية :

```
The average is 4.1800000E+00
i= 1    x= 3.0000000E+00    d= -1.1800000E+00
i= 2    x= -2.0000000E+00    d= -6.1800000E+00
i= 3    x= 1.2000000E+01    d= 7.8200000E+00
i= 4    x= 4.4000000E+00    d= 2.2000000E-01
i= 5    x= 3.5000000E+00    d= -6.8000000E-01
```

ويمكن التوسع في بعض التطبيقات باستخدام فهرس من النوع المتعدد كما هو موضح في المثال التالي .

### مثال (٩-٥)

فيما يلي جزءاً من برنامج بسكال يعرف منظومة لها بعد واحد ، ومن النوع الصحيح ، ولها فهرس من النوع المتعدد .

```
TYPE color = (red,white,blue,yellow,green);
VAR index : color;
    values : ARRAY [color] OF integer;
```

لاحظ أن القيم معرفة بأنها منظومة لها خمسة عناصر ، وكل عنصر يمثل قيمة صحيحة .

إذا ما أردنا كتابة قيمة كل عنصر ، فيمكن أن ندخل المكون التالي في مكان ما في البرنامج .

```
FOR index := red TO green DO writeln(values[index]);
```

ويمكن لعناصر المنظومة أن تكون من النوع المتعدد أيضاً كما هو موضح في المثال التالي .

### مثال (٩-٦)

فيما يلي انحرافاً عن الموقف المذكور في مثال ٩ - ٥ . افترض أن values هي منظومة بها 100 عنصر من النوع color . وعلى هذا ... فيمكن أن يحتوى برنامج البسكال على نوع التعريف وتوضيح المتغير التاليين :

```
TYPE color = (red,white,blue,yellow,green);
VAR values : ARRAY [1..100] OF color;
```

وكل عنصر من عناصر المنظومة يأخذ الآن قيمة واحدة من القيم red , green , yellow , blue , white . وقيم الفهرس تكون صحيحة ، وتتراوح من 1 إلى 100 .

هناك نوع آخر من التطبيقات يظهر بصورة متكررة ، محتوياً على استخدام السلاسل . في مثل هذه المواقف عادة ما تمثل السلسلة على أنها منظومة ذات بعد واحد من النوع الحرفي . وهذه الطريقة موضحة في المثال التالي .

### مثال (٩-٧)

الاسماء والعناوين ، فيما يلي برنامج بسكال بسيطاً يسمح بإدخال الاسم والعنوان ، وتخزينهما داخل الكمبيوتر ، ثم كتابتهما بعد ذلك . نفترض أن كل من الاسم واسم الشارع والمدينة المناظرة لانتدعى 60 خانة . وعلى

هذا ... فإننا نقدم ثلاث منظومات ، كل عنصر من عناصرها مكون من 60 خانة من النوع الحرفي . وأسمائها هي name , city , street على التوالي . ولا تحتاج أى سلسلة على أية حال لأن تحتوى على 60 خانة . بدلا من ذلك سوف ندخل رموزا متتالية داخل كل منظومة ، حتى نصل إلى نهاية السطر . وتستخدم دالة بوليان القياسية eoln فى هذا الغرض . وعلى هذا ... فإننا نستمر فى قراءة الرموز من كل سطر حتى تعود الدالة eoln بقيمة true . ( لاحظ أن eoln تصبح false عند بداية سطر جديد ) .

سوف نعد أيضا عدد الرموز فى كل منظومة ، بحيث إننا نعرف عدد الرموز ( عناصر المنظومة ) التى تكتب فيما بعد فى البرنامج . دعنا نشير إلى عدادات هذه الرموز بأنها citycount , namecount , streetcount على التوالي .

وفيما يلي محتويات برنامج البسكال . لاحظ أن البرنامج يحتوى بالضرورة على إجرائين : readin الذى يتسبب فى إدخال بيانات المدخلات داخل الكمبيوتر ، و writeout الذى يخرج المخرجات . لاحظ الملقنات للاسم والشارح والمدينة الموجودة داخل readin .

```
PROGRAM namesandaddresses(input,output);

(* THIS PROGRAM READS IN AND THEN WRITES OUT
   A NAME AND ADDRESS (STREET AND CITY) *)

VAR count,namecount,streetcount,citycount : 0..60;
    name,street,city : ARRAY [1..60] OF char;

PROCEDURE readin;
(* Read a name and address into the computer *)
BEGIN
    write('Name: ');
    count := 0;
    REPEAT
        count := count + 1;
        read(name [count])
    UNTIL eoln;
    namecount := count;
    readln;

    write('Street: ');
    count := 0;
    REPEAT
        count := count + 1;
        read(street [count])
    UNTIL eoln;
    streetcount := count;
    readln;

    write('City: ');
    count := 0;
    REPEAT
        count := count + 1;
        read(city [count])
    UNTIL eoln;
    citycount := count;
    readln
END; (* readin *)
```

(تكملة البرنامج فى الصفحة التالية)

```

PROCEDURE writeout;
(* Write a name and address out of the computer *)
BEGIN
  FOR count := 1 TO namecount DO write(name [count]);
  writeln;
  FOR count := 1 TO streetcount DO write(street [count]);
  writeln;
  FOR count := 1 TO citycount DO write(city [count]);
END; (* writeout *)

BEGIN (* main action block *)
  readin;
  writeln;
  writeout
END.

```

وينتج عن تنفيذ هذا البرنامج حوار المدخلات التالي ( استجابات المستخدم موجود تحتها خط ) .

Name: Susan H. Gottfried  
 Street: 129 Old Suffolk Drive  
 City: Monroeville, PA 15146

وبعد الانتهاء من إدخال البيانات ، فإن البرنامج يكتب الاسم والشارع والمدينة على النحو التالي :

Susan H. Gottfried  
 129 Old Suffolk Drive  
 Monroeville, PA 15146

وأخيرا يجب أن يميز القارئ أنه يمكن استخدام هذا البرنامج كنقطة بداية للعديد من البرامج الأخرى الأكثر تعقيدا . فمثلا يتطلب العديد من تطبيقات الأعمال ( مثل قوائم البريد والرواتب وحسابات المدينين ) أسماء وعناوين وبيانات مرتبطة بها لإدخالها الكمبيوتر وتعديلها بطريقة معينة ، ثم كتابتها بعد ذلك .

### مثال (٨-٩)

تسجيل قائمة بالأعداد . اعتبر المشكلة المشهورة بإعادة ترتيب قائمة بها  $n$  عدد من الأعداد في ترتيب لقيمتها الجبرية المتزايدة . ويكتب البرنامج بطريقة لا يستخدم فيها تخزين كبير .

وعلى هذا ... يحتوى البرنامج على منظومة واحدة فقط ذات بعد واحد من النوع الحقيقي ، اسمها  $x$  ، والتي تسجل عنصرا واحدا في نفس الوقت .

وتبدأ العملية بفحص محتويات المنظومة لتحديد أقل عنصر ، ثم استبدال هذا العنصر بأول عدد من أعداد المنظومة . ( عند ذلك يصبح أصغر عنصر في بداية القائمة ) وبعد ذلك يتم فحص بقية الأعداد ، وعددها  $n-1$  لتحديد أصغرها ، والذي يستبدل مع العدد الثاني في القائمة وهكذا ، حتى يعاد ترتيب محتويات المنظومة . ويتطلب ذلك عدد  $n-1$  من المسارات خلال المنظومة ، بالرغم من أن طول كل عملية فحص يقل بصفة دائمة من مسار المسار التالي له .

ولكى يحدد أقل عدد في كل مسار ، فإننا نقارن كل عدد في المنظومة  $x[i]$  على التوالي مع عدد البداية  $x[100]$  ، حيث 100 هو متغير صحيح يستخدم في تسنيفة tag أحد عناصر المنظومة . إذا كان  $x[i] < x[100]$  ، فإننا نستبدل العنصرين معا ، وإلا فإننا نترك العددين في موضعهما الأصلي . وبعد تطبيق هذه العملية على محتويات المنظومة ، يصبح أول عدد هو أصغر محتويات المنظومة ، ثم نكرر بعد ذلك هذه العملية عدد  $n-2$  من المرات ، ليصبح إجمالي عدد المسارات  $n-1$  ( $loc = 2, 1, \dots, n-1$ ) .

والسؤال المتبقى الوحيد هو كيفية تبديل عنصرين . نخزن أولا تخزينا مؤقتيا قيمة  $x[loc]$  في دليل للمستقبل ، ثم نحدد القيمة الحالية للمتغير  $x[i]$  في  $x[loc]$  . وأخيرا نحدد القيمة الأصلية original value للمتغير  $x[loc]$  التي سبق تخزينها في  $x[i]$  . وعند ذلك يكون التبادل قد حدث .

ويمكن تنفيذ ذلك في إجراء بسكال يسمى interchange كما هو موضح أدناه :

```
PROCEDURE interchange;
(* Interchange array elements from smallest to largest *)
BEGIN
  FOR loc := 1 TO n-1 DO
    FOR i := loc + 1 TO n DO
      IF x [i] < x [loc] THEN
        BEGIN
          temp := x [loc];
          x [loc] := x [i];
          x [i] := temp
        END
      END;
    END;
```

نفترض في هذا الإجراء أن  $loc$  ,  $i$  متغيرات صحيحة تتراوح قيمتها من 1 إلى 100 . كما نفترض أيضا أن  $temp$  متغير حقيقي يستخدم في التخزين المؤقت للمتغير  $x[loc]$  . لاحظ أن الإجراء يستخدم مكون FOR - TO متداخل .

دعنا نعتبر الآن عملية البرنامج الشامل كما هي موضحة في التخطيط التالي :

- (١) اقرأ حجم المنظومة  $x$  داخل الكمبيوتر .
- (٢) اقرأ عناصر المنظومة داخل الكمبيوتر ، وذلك ردا على ملقنات التداخل .
- (٣) أعد ترتيب الأعداد داخل المنظومة بالاتصال بإجراء interchange .
- (٤) اكتب القيم التي أعيد ترتيبها .

وفيما يلي برنامج البسكال .

```
PROGRAM reorder(input,output);
(* THIS PROGRAM REORDERS A ONE-DIMENSIONAL ARRAY
   OF REAL NUMBERS FROM SMALLEST TO LARGEST *)
VAR n,i,loc : 1..100;
    x : ARRAY [1..100] OF real;
    temp : real;
PROCEDURE interchange;
(* Interchange array elements from smallest to largest *)
BEGIN
  FOR loc := 1 TO n-1 DO
    FOR i := loc + 1 TO n DO
      IF x [i] < x [loc] THEN
        BEGIN
          temp := x [loc];
          x [loc] := x [i];
          x [i] := temp
        END
      END;
```

(تكملة البرنامج في الصفحة التالية)



```

END;  (* interchange *)
BEGIN  (* main action block *)
  write('How many numbers are there? ');
  readln(n);
  writeln;
  FOR i := 1 TO n DO
    BEGIN
      write('x [',i:3,']= ? ');
      readln(x [i])
    END;
  interchange;
  writeln;
  writeln('Rearranged Data:');
  writeln;
  FOR i := 1 TO n DO
    writeln('x [',i:3,']= ',x [i]:4:1);
  END.

```

ويبدأ تنفيذ هذا البرنامج بإدخال متداخل للبيانات ، مثل الموجود أدناه ، مع وضع خط تحت استجابة المستخدم .

```

How many numbers are there? 5
x [ 1] = ? 4.7
x [ 2] = ? -2.3
x [ 3] = ? 12.9
x [ 4] = ? 8.8
x [ 5] = ? 6.0

```

يعيد البرنامج ترتيب الكميات الخمس المدخلة ، وينتج المخرجات التالية :

```

x [ 1] = -2.3
x [ 2] = 4.7
x [ 3] = 6.0
x [ 4] = 8.8
x [ 5] = 12.9

```

## 2. MULTIDIMENTICNAL ARRAYS

### ٢ - منظومات متعددة الأبعاد :

يجب ألا يقتصر مفهوم المنظومة على منظومات أحادية البعد في البسكال ، فيمكن تعريف منظومات متعددة الأبعاد أيضا . وفي واقع الأمر ... مثل هذه المنظومات مفيدة جدا في أنواع عديدة ومختلفة من التطبيقات .

لقد رأينا بالفعل أنه يمكن التفكير في منظومة ذات بعد واحد كقائمة ( أى عمود واحد ) من عناصر البيانات . وتقدم المنظومة متعددة الأبعاد التوسع الطبيعي لهذه الفكرة . وعلى هذا ... فيمكننا أن نفكر في منظومة مزدوجة البعد على أنها جدول من عناصر البيانات يحتوى على صفوف وأعمدة . وأول بعد ( أو أول فهرس ) يشير إلى رقم السطر ، أما الثانى بعد ( أو ثانى فهرس ) فيشير إلى رقم العمود . وبالمثل يمكن التفكير في المنظومة ذات ثلاثة أبعاد بأنها مجموعة من الجداول مثل صفحات الكتاب .

افرض على سبيل المثال أن table هو منظومة ذات بعدين تحتوى على m صف و n عمود . العناصر الفردية للمنظومة هي :

[1,1], table [1,2], . . . , table [1, n], table [2,1],  
table [2,2], . . . , table [2, n], . . . , table [m, 1],  
table [m, 2], . . . , table [m, n]

وعلى هذا ... فيمكن رؤية محتويات المنظومة بالطريقة التالية :

$$\text{table} = \begin{bmatrix} \text{table [1,1]} & \text{table [1,2]} & . . . & \text{table [1,n]} \\ \text{table [2,1]} & \text{table [2,2]} & . . . & \text{table [2,n]} \\ \text{table [3,1]} & \text{table [3,2]} & . . . & \text{table [3,n]} \\ . & . & . & . \\ \text{table [m,1]} & \text{table [m,2]} & . . . & \text{table [m,n]} \end{bmatrix}$$

ويغض النظر عن أبعاد المنظومة ، فدائماً ماتحتوى المنظومة على مجموعة من عناصر البيانات من نفس النوع . ويمكن الإشارة إلى كل عناصر البيانات بمعرف واحد ( أى باسم مشترك للمنظومة ) . ويمكن الإشارة إلى عنصر فردى فى المنظومات ( أى عنصر بيانات فردى ) بتحديد اسم المنظومة ، تليه القيم المناسبة للفهارس ( أو للدلالة ) محصورة بين أقواس مربعة ومفصولة عن بعضها بفواصل .

مثال (٩ - ٩)

يحتوى أحد برامج البسكال على منظومة ذات بعدين من النوع الحقيقى اسمها table . تمثل هذه المنظومة جدولاً من الأعداد يحتوى على 60 صفاً ، و 150 عموداً كحد أقصى . وعلى هذا ... فإن أول فهرس يتراوح من 1 إلى 60 بينما يتراوح الفهرس الثانى من 1 إلى 150 .

افرض أننا نريد الاتصال بعنصر المنظومة الموجود فى الصف الثالث والعمود السابع . يمكن الإشارة إلى هذا العنصر ببساطة بكتابة table [3,7] . وبالمثل يمكننا الإشارة إلى عنصر المنظومة الموجود فى الصف ١ والعمود ٧ بأنه table [١, ٧] ، حيث ١ متغيرات صحيحة النوع تستخدم كفهارس . وعلى هذا ... إذا ما حددت 12 كقيمة للمتغير ١ ، وحددت 5 كقيمة للمتغير ٧ ، فإن table [١,٧] تشير إلى عنصر المنظومة الموجود فى العمود الخامس والصف الثانى عشر .

ويمكن ظهور تعريف المنظومة متعددة الأبعاد مع توضيح المتغيرات بنفس طريقة المنظومة ذات البعد الواحد . وعلى أية حال ... يجب تحديد فهرس منفصل لكل بعد من أبعاد المنظومة . وعلى هذا ... يمكن كتابة تعريف المنظومة كمايلي :

VAR array name : ARRAY [index 1 type, index 2 type, . . . ,  
index n type] OF type

يمكن أن تكون أنواع الفهارس الترتيبية بسيطة ( أى صحيحة أو حرفية و بوليان أو متعددة ) أو من نوع المدى الجزئى . وليس هناك حاجة لأن تكون جميعها من نفس النوع ، إلا أن عناصر البيانات نفسها فى المنظومة يجب أن تكون من نفس النوع ، بما فى ذلك الأنواع المرتبة .

مثال ، (٩-١٠)

اعتبر مرة أخرى المنظومة الحقيقية ذات البعدين المسماة table المذكورة فى المثال السابق . يمكن تعريف هذه المنظومة بالطريقة التالية :

```
VAR table : ARRAY [1..60, 1..150] OF real;
```

وفيما يلى طريقة أخرى للتعبير عن نفس التعريف :

```
TYPE index1 = 1..60;
      index2 = 1..150;
VAR table : ARRAY [index1,index2] OF real;
```

وفيما يلى طريقة ثالثة للتعبير عن نفس التعريف :

```
CONST limit1 = 60;
      limit2 = 150;
TYPE index1 = 1..limit1;
      index2 = 1..limit2;
VAR table : ARRAY [index1,index2] OF real;
```

ومن الواضح أن التعريف الأول هو أبسط هذه التعريفات .

لاحظ أن كل من الفهرسين من نوع المدى الجزئى الصحيح ، إلا أن عناصر المنظومة نفسها من النوع الحقيقى .

ويتطلب العديد من التطبيقات التى تحتوى على استخدام منظومات متعددة الأبعاد دورات متداخلة ، بمعدل دورة لكل بعد من أبعاد المنظومة . وعلى هذا ... فالتطبيق الموجة ناحية تشغيل العناصر فى جدول يمكن أن يستخدم بورتين ، واحدة داخل الأخرى . ويمكن استخدام الدورة الخارجية فى تشغيل كل صف من الصفوف الفردية ، والدورة الداخلية فى تشغيل الأعمدة داخل كل صف . وطريقة تحقيق ذلك موضحة فى المثال التالى .

مثال (٩-١١)

معاملة الجدول . دعنا نكتب الآن برنامج بسكال متداخلاً ، يقرأ جدولاً من أعداد حقيقية داخل الكمبيوتر ، ويخزنها فى منظومة ذات بعدين ، ويحسب مجموعة الأعداد الموجودة فى كل صف وفى كل عمود ، ثم يكتب الجدول ومعه مجموع الصفوف ومجموع الأعمدة التى تم حسابها .

table = منظومة حقيقية ذات بعدين ، تحتوى على جدول معطى ، والمجموع الذى تم حسابه .

nrows = متغير صحيح يحدد عدد الصفوف الموجودة فى الجدول .

`ncols` = متغير صحيح يحدد عدد الأعمدة في الجدول .

`row` = عداد صحيح يحدد رقم الصف .

`col` = عداد صحيح يحدد رقم العمود .

نفرض أن حجم الجدول لن يتعدى ١٠ صفوف و ١٠ أعمدة . ونضيف صفًا إضافيًا وعمودًا إضافيًا داخل المنظومة على أية حال ، وذلك لتخزين مجموع كل صف ومجموع كل عمود فيهما . وعلى هذا ... يوضع مجموع كل صف في العمود الموجود على أقصى اليمين ( أى العمود رقم `ncols+1` ) ، ويوضع مجموع كل عمود في الصف الموجود في قاعدة الصفوف ( أى الصف رقم `nrows+1` ) . وعلى هذا ... فإننا نعرف `table` بأنه منظومة حقيقية ذات بعدين ، لها حد أقصى من عدد الصفوف ١١ ، ومن عدد الأعمدة ١١ .

دعنا نستخدم تكوين الأجزاء في إعداد البرنامج . دعنا بصفة خاصة نكتب إجراءات منفصلة لقراءة البيانات داخل الكمبيوتر ، وحساب مجموع كل صف ومجموع كل عمود وكتابة الجدول النهائي . وهذه الإجراءات تسمى `readinput` , `rowsums` , `columnsums` , `writeoutput` على التوالي .

والمنطق المطلوب لكل إجراء واضح ومباشر ، بالرغم من أنه يجب ملاحظة أنه مطلوب دورة مزدوجة داخل كل إجراء من هذه الإجراءات . فلقراءة الجدول الأصلي داخل الكمبيوتر على سبيل المثال يجب أن تقدم الدورة المزدوجة التالية .

```
FOR row := 1 TO nrows DO
  BEGIN
    FOR col := 1 TO ncols DO read(table [row,col]);
    writeln
  END
```

( العبارة المكتوبة مطلوبة لكي ينتقل إلى السطر التالي ) . وبالمثل الدورة التالية مطلوبة لحساب مجموع كل صف .

```
FOR row := 1 TO nrows DO
  BEGIN
    table [row,ncols+1] := 0;
    FOR col := 1 TO ncols DO
      table [row,ncols+1] := table [row,ncols+1] + table [row,col]
    END
```

ويمكن استخدام مكونات نورات مزدوجة أخرى لحساب مجموع كل عمود ، وكتابة الجدول النهائي .

وفيما يلي برنامج بسكال كاملاً لأداء ذلك :

```
PROGRAM table1(input,output);

(* THIS PROGRAM READS IN A TABLE OF NUMBERS,
   SUMS THE COLUMNS WITHIN EACH ROW,
   AND THEN SUMS THE ROWS WITHIN EACH COLUMN *)

VAR row,col : 1..11;
    nrows,ncols : 1..10;
    table : ARRAY [1..11, 1..11] OF real;
```

(تكملة البرنامج في الصفحة التالية)

```

PROCEDURE rowsums;
(* Add the columns within each row *)
BEGIN
  FOR row := 1 TO nrows DO
    BEGIN
      table [row,ncols+1] := 0;
      FOR col := 1 TO ncols DO
        table [row,ncols+1] := table [row,ncols+1] + table [row,col]
      END
    END;
  (* rowsums *)

PROCEDURE columnsums;
(* Add the rows within each column *)
BEGIN
  FOR col := 1 TO ncols DO
    BEGIN
      table [nrows+1,col] := 0;
      FOR row := 1 TO nrows DO
        table [nrows+1,col] := table [nrows+1,col] + table [row,col]
      END
    END;
  (* columnsums *)

PROCEDURE readinput;
(* Read the elements of the table *)
BEGIN
  write(' How many rows? (1..10) ');
  readln(nrows);
  writeln;
  write(' How many columns? (1..10) ');
  readln(ncols);
  writeln;
  FOR row := 1 TO nrows DO
    BEGIN
      writeln(' Enter data for row no. ',row:2);
      FOR col := 1 TO ncols DO read(table [row,col]);
      writeln
    END
  END;
  (* readinput *)

PROCEDURE writeoutput;
(* Write out the table and the corresponding sums *)
BEGIN
  writeln(' Original table, with row sums and column sums:');
  writeln;
  FOR row := 1 TO nrows + 1 DO
    BEGIN
      FOR col := 1 TO ncols + 1 DO write(table [row,col]:6:1);
      writeln
    END
  END;
  (* writeoutput *)

BEGIN (* main action block *)
  readinput;
  rowsums;
  columnsums;
  writeoutput
END.

```

افترض أن البرنامج يستخدم أعداد الجداول التالية في التشغيل .

2.5	-6.3	14.7	4.0
10.8	12.4	-8.2	5.5
-7.2	3.1	17.7	-9.1

عند تنفيذ البرنامج يحدث الحوار التالي ( استجابة المستفيد موضوع تحتها خط )

How many rows? (1..10) 3

How many columns? (1..10) 4

Enter data for row no. 1

2.5 -6.3 14.7 4.0

Enter data for row no. 2

10.8 12.4 -8.2 5.5

Enter data for row no. 3

-7.2 3.1 17.7 -9.1

عند ذلك يحسب البرنامج مجموع كل صف ، ثم ينتج المخرجات التالية :

2.5	-6.3	14.7	4.0	14.9
10.8	12.4	-8.2	5.5	20.5
-7.2	3.1	17.7	-9.1	4.5
6.1	9.2	24.2	0.4	0.0

ويمكننا أن نرى من هذه المخرجات أن مجموع عناصر الصف الأول هي 14.9 ( وهي 2.5-6.3+14.7+4 ) ويمكننا أن نرى أن مجموع عناصر العمود الأول هي 6.1 ( وهي 2.5+10.8-7.2 = 6.1 ) ويمكننا

ولا تحتاج قهارس المنظومة متعددة الأبعاد أن تكون كلها من نفس النوع . والقيد الوحيد هو أن كل فهرس يكون من النوع الترتيبي البسيط .

مثال (٩-١٢)

فيما يلي بعض توضيحات إضافية لمنظومة متعددة الأبعاد :

```
TYPE color = (red,white,blue,yellow,green);
index = 1..100;
VAR sample : ARRAY [color,index] OF boolean;
```

وعلى هذا ... فأحد عناصر sample يمكن أن يكون sample [white,25] ، وعنصر آخر يمكن أن يكون sam-ple [green,66] . وحيث إن sample هي منظومة من نوع بولياني ، فإن القيم المحددة للعناصر الفردية تكون بولياني أيضا ، أي أن قيمتها يجب أن تكون صحيحة true أو خاطئة false فقط .

تذكر أن المنظومة يمكن أن يكون لها أي نوع ، بما في ذلك النوع المرتب . وعلى هذا ... فمن الممكن تعريف منظومة ذات بعد واحد وعناصرها نفسها تكون عبارة عن منظومات ذات بعد واحد ( أي أن كل العناصر لها نفس البعد ) ومن نفس النوع ، مثل المنظومة الأم . ونتيجة تعريف مثل هذه المنظومة تكافئ منظومة ذات بعدين معتادة .

### مثال (٩-١٣)

دعنا نعرف منظومة حرفية ذات بعد واحد تسمى codes ، عناصرها عبارة عن منظومات حرفية ذات بعد واحد أيضا . نفرض أن codes بها 25 عنصرا كحد أقصى ، وأن كل عنصر عبارة عن منظومة بها 50 عنصرا ، يمكن كتابة تعريف المنظومة كما يلي :

```
VAR codes : ARRAY [1..25] OF ARRAY [1..50] OF char;
```

وهذا يكافئ تعريف منظومة حرفية ذات بعدين معتادة ، وعلى هذا ... فيمكننا كتابة تعريفها بطريقة بديلة كما يلي :

```
VAR codes : ARRAY [1..25, 1..50] OF char;
```

كل توضيح من هذين التوضيحين يعرف جنوفاً به 25 صفّاً على الأكثر ، وفي كل صف ما لا يزيد عن 50 عنصراً من النوع الحرفي .

العناصر الفردية للمنظومة يمكن الاتصال بها بطريقتين مختلفتين أيضا . فمثلا يمكن الاتصال بالعنصر الموجود في الصف i والعمود j كما يلي :

```
code [i][j]
```

أو كما يلي :

```
code [i,j]
```

وبالمثل يمكن الاتصال بالصف j كما يلي :

```
code [i]
```

وعلى هذا ... فالعنصر الموجود في الصف 12 والعمود 20 يمكن الاتصال به بكتابة code[12][20] ، أو كتابة code[12,20] . كما أن كل عنصر من الصف 12 يمكن الاتصال بها بكتابة code[12] أيضا .

وفي معظم التطبيقات يكون من المريح استقلال عناصر منظومة متعددة الأبعاد باستخدام الطريقة الأصلية كما هي مذكورة في مثال ٩ - ١١ ، بدلا من الطريقة سالفة الذكر .

### ٣ - عمليات تجرى على محتويات المنظومات : 3. OPERATIONS WITH ENTIRE ARRAYS

هناك عمليات معينة يمكن إجراؤها على محتوى المنظومة ، وتؤثر - على ذلك - على كل عناصر المنظومة بنفس الطريقة . يمكننا بصفة خاصة تعريف أنواع المنظومات وتحديد منظومات لمنظومات أخرى ، وتمرير منظومات كمؤشرات إلى إجراءات أو إلى دوال . دعنا نعتبر كل من هذه المعالم بشئ من التفصيل .

تستخدم بعض البرامج العديد من المنظومات المختلفة من نفس النوع ، ولها نفس الأبعاد ، وفي مثل هذه الحالات يكون من المريح تعريف نوع واحد من أنواع المنظومات ، ثم توضيح المنظومات الفردية كمتغيرات من هذا النوع .

وفي كلمات أخرى ... يمكن التعبير عن تعريف نوع المنظومة كما يلي :

```
TYPE name = ARRAY [index 1 type, index 2 type, . . . ,
                    index n type] OF type
```

حيث إن أنواع الفهرس ونوع البيانات المنظومة لها نفس المعنى ، كما في توضيح المتغير بأنه من نوع المنظومة ( انظر القسم رقم ٢ ) .

مثال (٩-١٤)

افرض أن group 1 و group 2 هما منظومتان من النوع الصحيح ، لهما بعدان ، وكل منهما به 25 صفًا كحد أقصى ، و 50 عمودًا كحد أقصى . يمكن تعريف هاتين المنظومتين بالطريقة التالية :

```
TYPE table = ARRAY [1..25, 1..50] OF integer;
VAR group1, group2 : table;
```

وعلى هذا ... فإن table يعرفه المستفيد بأنه من النوع المرتب ( منظومة صحيحة النوع ، ذات بعدين ، لها 25 صفًا كحد أقصى ، و 50 عمودًا كحد أقصى ، وذلك بالتحديد ) . ويوضح المتغيران group 1 و group 2 بأنهما من نفس نوع table . وعلى هذا ... فإن كل من group 1 و group 2 مصفوفة من النوع الصحيح ، ذات بعدين ، ولها 25 صفًا كحد أقصى ، و 50 عمودًا كحد أقصى .

ولاحتياج توضيحات المتغيرات أن يكون لها نفس مدى تعريف النوع . فمثلا يمكن تعريف النوع داخل المجموعة الرئيسية ، ويمكن أن يظهر توضيحات لمتغير منظومة معين فوراً بعد ذلك . وتوضيحات المتغير المتبقية يمكن عندئذ أن تظهر في إجراء أو في دالة .

وفي بعض الأحيان يكون من المرغوب فيه تحديد كل عناصر منظومة واحدة لتناظر عناصر منظومة أخرى ، حيث تكون كلا من المنظومتين من نفس النوع ، ولهما نفس الأبعاد . ويمكن تحقيق ذلك بعبارة تحديد واحدة ، مع تجنب الحاجة لتحديد كل عنصر على انفراد داخل أحد أنواع مكونات البورات .

مثال (٩-١٥)

اعتبر المنظومتين الصحيحتين group 1 و group 2 مرة أخرى ، والمعرفتين في المثال السابق . افرض أن كل عناصر المنظومة group 1 تم قراءتها بالفعل داخل الكمبيوتر ، وأنتا ترغب في تحديد هذه العناصر لتناظر عناصر المنظومة group 2 . يمكن تحقيق ذلك ببساطة بكتابة مايلي :

```
group2 := group1;
```

لاحظ أن عبارة التحديد الفردية هذه تكافئ الدورة المزدوجة المتداخلة التالية :

```
FOR row := 1 TO 25 DO
  FOR col := 1 TO 50 DO
    group2 [row,col] := group1 [row,col];
```



ومن الواضح أن عبارة التحديد البسيطة أبسط كثيرا .

وعلى أية حال ... ليس من الممكن أن تحتوى تعبيرات عديدة أو بوليان على كل محتويات المنظومة . وعلى هذا ... فإذا كانت c,b,a متغيرات منظومة ، فإن مثل العبارة التالية غير مسموح به .

$$c := a + b;$$

وبالمثل لا يمكن قراءة جميع محتويات المنظومة داخل الكمبيوتر بعبارة read أو readln واحدة . وعلى هذا ... يكون مطلوبا مكون دورة لقراءة العناصر المتعددة للمنظومة . وبصفة عامة ... فإن نفس القيد يقع على كتابة محتويات المنظومة كلها بعبارة write أو عبارة writeln واحدة . سوف نرى استثناء هاما لهذا القيد الأخير فى قسم ه من الفصل التاسع .

ويمكن تمرير المحتويات الكلية للمنظومة إلى إجراء أو إلى دالة كمؤشر . لاحظ على أية حال أن المؤشر الرسمى المناظر ( داخل الإجراء أو الدالة ) يجب أن يصاحبه نوع بيانات مميز . وفى هذه الحالة يكون النوع منظومة . وعلى هذا يجب أن يعرف نوع المنظومة صراحة داخل مجموعة تحتوى على الإجراء أو على الدالة .

والطريقة المريحة لتحقيق هذا المتطلب هى تعريف نوع منظومة شامل global array type كما سبق ذكره فى بداية هذا القسم . ويمكن بعد ذلك توضيح مؤشرات نوع المنظومة الفعلية والمؤشرات الرسمية المناظرة كأعداد لهذه المنظومة كما هو مطلوب فى البرنامج . وهذه الطريقة موضحة فى المثال التالى .

### مثال (٩-١٦)

جمع جدولين من الأعداد . افترض أننا نريد قراءة جدولين صحيحين داخل الكمبيوتر ، ونحسب مجموع العناصر المناظرة ، أى :

$$c[i,j] = a[i,j] + b[i,j]$$

ثم نكتب بعد ذلك جدولا جديدا يحتوى على هذه المجموعات . نفرض أن كل الجداول تحتوى على نفس عدد الصفوف والأعمدة ، والتى لا تتعدى 20 صفًا و 30 عمودًا .

دعنا نستخدم تعريفات المتغيرات التالية :

table = منظومة ذات بعدين ، صحيحة النوع ، لها 20 صفًا كحد أقصى ، و 30 عمودًا كحد أقصى .

a , b , c = منظومات ، كل منها له بعدان من نفس نوع المنظومة table .

nrows = متغير صحيح يحدد العدد الفعلى للصفوف فى كل جدول .

ncols = متغير صحيح يحدد العدد الفعلى للأعمدة فى كل جدول .

row = عداد صحيح يحدد رقم الصف .

col = عداد صحيح يحدد رقم العمود .

سوف نستخدم تكوين أجزاء تشبه التكوين المستخدم في مثال ١١ - ٩ ، وبصفة خاصة فإننا نعرف إجراء لقراءة البيانات داخل أى جدول ، وإجراء آخر لتجميع كل العناصر ، وإجراء ثالث لكتابة جدول جديد يحتوى على المجموع المحسوب . دعنا نسمى هذه الإجراءات readinput , computesums , writeoutput على التوالى .

والبرنامج الفعلى يكون مباشرا كما هو مبين أدناه :

```
PROGRAM table2(input,output);

(* THIS PROGRAM READS IN TWO TABLES OF INTEGERS,
   CALCULATES THE SUMS OF THEIR RESPECTIVE ELEMENTS,
   AND WRITES OUT THE RESULTING TABLE CONTAINING THE SUMS

TYPE table = ARRAY [1..20,1..30] OF integer;
VAR a,b,c : table;
    row,nrows : 1..20;
    col,ncols : 1..30;

PROCEDURE readinput(VAR t : table);
(* Read the elements of one table *)
BEGIN
    FOR row := 1 TO nrows DO
        BEGIN
            writeln(' Enter data for row no. ',row:2);
            FOR col := 1 TO ncols DO read(t[row,col]);
            writeln
        END
    END (* readinput *)

PROCEDURE computesums(t1,t2 : table; VAR t3 : table);
(* Sum the elements of two similar tables *)
BEGIN
    FOR row := 1 TO nrows DO
        FOR col := 1 TO ncols DO
            t3[row,col] := t1[row,col] + t2[row,col]
        END
    END; (* computesums *)

PROCEDURE writeoutput(t : table);
(* Write out the elements of a table *)
BEGIN
    writeln(' Sums of the elements:');
    writeln;
    FOR row := 1 TO nrows DO
        BEGIN
            FOR col := 1 TO ncols DO write(t[row,col]:4);
            writeln
        END
    END; (* writeoutput *)

BEGIN (* main action block *)
    write(' How many rows? (1..20) ');
    readln(nrows);
    writeln;
    write(' How many columns? (1..30) ');
    readln(ncols);
    writeln;
```

(تكملة البرنامج فى الصفحة التالية)

```

writeln(' First table:');
writeln;
readinput(a);
writeln(' Second table:');
writeln;
readinput(b);
computesums(a,b,c);
writeoutput(c)
END.

```

يحتوى هذا البرنامج على خواص عديدة يجب ملاحظتها . لاحظ أولا أن البرنامج يحتوى على تعريف منظومة تسمى table ، كما أن هناك توضيحا لثلاثة متغيرات a , b , c داخل المجموعة الرئيسية على أنها منظومات من نفس نوع table ، بالإضافة إلى ذلك تحتوى الإجراءات على مؤشرات من نوع المنظومة ، موضحة أيضا بأنها من نفس نوع table . لاحظ أن مؤشر نوع المنظومة t ، والموجود فى الإجراء readinput ، هو مؤشر متغير variable . وهذا يسمح لعناصر t أن تعود من الاجراء إلى المجموعة الرئيسية . ومن ناحية أخرى ... فإن متغير نوع المنظومة t الموجود فى writeoutput هو مؤشر قيمة value ، حيث إن عناصر t المنقولة إلى هذا الإجراء ليست فى حاجة إلى العودة إلى المجموعة الرئيسية .

الإجراء المتبقى computersums يستغل كل من مؤشرات القيمة من نوع المنظومة ، ومؤشرات المتغير من نوع المنظومة . وبصفة خاصة فإن أول مؤشرين ( t1 , t2 ) هما مؤشرا قيمة ، حيث إنهما يمثلان بيانات مدخلات للإجراء . أما المؤشر الثالث ( t3 ) ، فهو مؤشر متغير ، حيث إنه يمثل معلومات يجب عودتها إلى المجموعة الرئيسية .

افرض الآن أن البرنامج يستخدم لجمع جدولى الأعداد التالين .

الجدول الثانى	الجدول الأول
10 11 12 13	1 2 3 4
14 15 16 17	5 6 7 8
18 19 20 21	9 10 11 12

ينتج عن تنفيذ البرنامج الحوار التالى أولا ، وذلك لإدخال البيانات ( مرة أخرى استجابات المستفيد موضوع تحتها خط ) .

How many rows? (1..20) 3

How many columns? (1..30) 4

First table:

Enter data for row no. 1

1 2 3 4

Enter data for row no. 2

5 6 7 8

Enter data for row no. 3

9 10 11 12

(تكملة البرنامج فى الصفحة التالية)

Second table:

Enter data for row no. 1

10 11 12 13

Enter data for row no. 2

14 15 16 17

Enter data for row no. 3

18 19 20 21

عند ذلك يجمع البرنامج كل العناصر ، وينتج المخرجات التالية .

Sums of the elements:

11	13	15	17
19	21	23	25
27	29	31	33

ومن السهل التأكد من صحة هذه القيم في الواقع .

#### 4. PACKED ARRAYS

#### ٤ - منظومات مضغوطة :

يمكن تعريف بعض المنظومات ، بحيث إنها تستغل ذاكرة الكمبيوتر بكفاءة ، وذلك بضغط عناصر البيانات لتكون قريبة من بعضها البعض . وتسمح هذه السمة بتخزين كمية معلومات كبيرة في كمية معينة من الذاكرة .

والكى يستغل ضغط المنظومة يجب استخدام الكلمات المحجوزة PACKED ARRAY في مواصفات النوع ، أى :

VAR array name : PACKED ARRAY [index 1 type ,  
index 2 type , . . . , index n type] OF type

وهذه السمة مرتفعة الكفاءة مع عناصر منظومة من النوع الحرفى ، ونوع بوليان ، والنوع المتعدد ، أو مع بيانات من نوع المدى الجزئى .

ومن ناحية أخرى ... فإن الوفر في التخزين الذى يتحقق من الضغط يمكن أن يتلشى بسبب فقدان سرعة الحسابات . وعلى هذا ... فأي برنامج يستغل منظومة مضغوطة يحتوى على موازنة بين هذين العنصرين . وليس من الواضح لبرنامج معين أى طريقة هي الأفضل . وهذا التمييز يتغير بصفة عامة من تطبيق لآخر .

وهناك ظروف معينة يكون فيها الضغط مقيدا ، وذلك من وجهة نظر اقتصاديات التخزين ، ومن وجهة نظر سرعة الحسابات أيضا . ويميل هذا لأن يكون صحيحا في البرامج التى تحتوى على عدد كبير من تحديدات منظومات ( أى تحديد العناصر من منظومة لأخرى ) أو استدعاء إجراء أو دالة تمر فيه المنظومات كمؤشرات قيمة .

مثال (٩-١٧)

اعتبر مرة أخرى المنظومة ذات البعدين ، ومن نوع بوليان المسماة sample ، والمعرفة في مثال ٩ - ١٧ . إذا

ما أوضحنا الآن sample بأنها منظومة مضغوطة ، فإن توضيح المنظومة يظهر كما يلي :

```
TYPE color = (red,white,blue,yellow,green);
  index = 1..100;
VAR sample : PACKED ARRAY [color,index] OF boolean;
```

سوف يتسبب هذا التوضيح في ضغط عناصر المنظومة sample في مساحة ذاكرة أقل عن التوضيح المناظر الموجود في مثال ٩ - ١٢.

يحتوي البسكال على إجرائين قياسيين ، هما : pack , unpack ، يستخدمان في تحويل المنظومات غير المضغوطة إلى منظومات مضغوطة ، والعكس صحيح . وبصفة خاصة ... فإن الإجراء القياسي pack ينقل بعض العناصر أو كلها ، الخاصة بمنظومة غير مضغوطة إلى منظومة مضغوطة . وهذا التكوين هو :

```
pack(unpacked array name , index , packed array name);
```

حيث يشير index إلى أول عنصر في المنظومة غير المضغوطة التي ستنتقل

وعند الاتصال بهذا الإجراء تستمر عملية النقل حتى تملأ كل عناصر المنظومة المضغوطة . ويجب أن تحتوي المنظومة غير المضغوطة ( منظومة المصدر ) على عناصر كافية بعد عنصر index ، بحيث إن كل العناصر داخل المنظومة المضغوطة ( منظومة الهدف ) يمكن ملؤها .

مثال (٩-١٨)

اعتبر المنظومتين التاليتين :

```
VAR unpacked : ARRAY [1..20] OF char;
    packed    : PACKED ARRAY [1..20] OF char;
```

العبارة التالية :

```
pack(unpacked,1,packed);
```

تتسبب في أن كل عنصر في unpack ( بدءاً بالموقع ١ ) ينقل إلى الموقع المناظر داخل pack . أي أن :

```
packed [1] := unpacked [1];
```

```
packed [2] := unpacked [2];
```

```
packed [20] := unpacked [20];
```

وينقل الإجراء القياسي unpack عناصر المنظومة المضغوطة إلى المنظومة غير المضغوطة ، بدءاً بالعنصر index من المنظومة غير المضغوطة . والتكوين هو :

```
unpack(packed array name , unpacked array name , index);
```

وتستمر عملية النقل حتى تنقل كل عناصر المنظومة المضغوطة . ويجب أن يكون في المنظومة غير المضغوطة ( أى منظومة الهدف ) عناصر كافية بعد العنصر index ، بحيث يمكن نقل كل العناصر داخل المنظومة المضغوطة ( منظومة المصدر ) .

مثال (٩-١٩)

اعتبر المنظومتين التاليتين :

```
VAR packed : PACKED ARRAY [1..13] OF char;
    unpacked : ARRAY [1..20] OF char;
```

افرض أن packed تحتوى على الرموز الثلاثة عشر التالية :

North America

وعلى هذا ... ينتج عن العبارة التالية :

```
unpack(packed,unpacked,5);
```

التحديدات التالية :

```
unpacked [5] := packed [1];    (hence unpacked [5] = 'N')
```

```
unpacked [6] := packed [2];    (hence unpacked [6] = 'o')
```

```
unpacked [17] := packed [13]; (hence unpacked [17] = 'a')
```

أما بقية عناصر unpacked الأخرى ، فتبقى بدون تعريف .

وهناك قيد واحد يجب أخذه في الاعتبار عند استخدام منظومات مضغوطة ، وهو أنه لا يمكن أن تمرر العناصر الفردية لمنظومة مضغوطة كمؤشرات فعلية إلى إجراءات أو نوال ، إلا أن المحتوى الكلى يمكن - على أية حال - تمريره بالطريقة المعتادة .

## ٥ - السلاسل ومتغيرات السلاسل : 9. STRINGS AND STRING VARIABLES

مفهوم المنظومة المضغوطة مهم بصفة خاصة بالاتصال مع السلاسل ، حيث إن السلسلة التى بها n رمزا عادة ماتعتبر بأنها منظومة ذات بعد واحد مضغوطة ، وبها n عنصر ، ولها النوع الحرفى .

مثال (٩-٢٠)

افرض أن أحد برامج البسكال يحتوى على تعريف الثابت التالى :

```
CONST title = 'The Super-Duper Computer Company';
```

يمثل المعرف title منظومة مضغوطة مكونة من 32 عنصرا من النوع الحرفي .

ويعتمد مفهوم السلسلة ليشمل متغيرات مثل الثوابت ، وعلى هذا ... فإن متغير من النوع :

PACKED ARRAY [1..n] OF char

يشار إليه بأنه متغير سلسلة string variable . مثل هذه المتغيرات يمكن استخدامها لتمثيل سلاسل لها طول n .

مثال (٩-٢١)

افرض أن برنامج بسكال يحتوى على توضيح المتغير التالي :

VAR heading : PACKED ARRAY [1..14] OF char;

المعرف heading هو متغير سلسلة ، حيث إنه يمثل منظومة حرفية مضغوطة بها 14 عنصرا .

ويشتمل البسكال القياسى على بعض اختيارات خاصة للمنظومات المضغوطة ، والتي يمكن استخدامها مع السلاسل أو متغيرات السلاسل فقط . فمثلا يمكن تحديد سلسلة بها n عنصرا لمتغير سلسلة ذى n عنصر ، وذلك باستخدام عبارة تحديد عادية . ( لاحظ أن هذه العملية الفردية تشمل تحديد لتسلسل عناصر السلسلة ) .

مثال (٩-٢٢)

اعتبر مرة أخرى المنظومة المضغوطة heading المعروفة في المثال السابق ، حيث إن heading هو متغير سلسلة له 14 عنصرا ، فيمكننا أن نحدد أى سلسلة لها 14 عنصرا للمنظومة heading ، أى :

heading := 'Dallas Cowboys';

ويمكننا أن نحدد بالطبع متغير سلسلة واحد لمتغير آخر ، على أن يشتمل كلا من المتغيرين على نفس عدد العناصر . ( تذكر أن هذه السمة تسرى على كل المنظومات كما سبق ذكره في القسم ٣ من هذا الفصل )

مثال (٩-٢٣)

افرض الآن أن برنامج البسكال يحتوى على توضيح المتغير التالي :

VAR factory,warehouse : PACKED ARRAY [1..11] OF char;

يستخدم المعرفان factory , warehouse متغيرات سلسلة ، حيث إن كلا منهما يمثل منظومة مضغوطة حرفية . لاحظ أيضا أن كلا من المتغيرين معرف بأنه منظومة لها 11 عنصرا . وعلى هذا ... يمكن تحديد إحدى المنظومات للمنظومة الأخرى .

والآن افترض أن البرنامج يحتوى على تحديد السلسلة التالية :

```
factory := 'Seattle, WA';
```

وعلى ذلك ... ففي نقطة لاحقة في البرنامج يمكننا أن نحدد نفس السلسلة لمتغير سلسلة آخر له ١١ عنصرا ،  
مثل :

```
warehouse := factory;
```

وعلى هذا ... فإن warehouse يمثل أيضا السلسلة "Seattle,WA." .

ويمكن مقارنة السلاسل ذات الطول المتساوى مع بعضها بواسطة المؤشرات العلاقية . ويمكن إجراء مثل هذه المقارنات بين ثابتى سلسلة أو متغيرى سلسلة ، أو ثابت سلسلة ومتغير سلسلة . وفي كل حالة من هذا الحالات فإننا ننتج تعبير بولياني يكون صحيحا true أو خاطئا false .

وعند استخدام مؤثر علاقى لتوصيل سلاسل أو متغيرات سلاسل ، فإن المؤثرين = و > يحددان التكافؤ وعدم التكافؤ على التوالي . وتشير المؤشرات الأربعة المتبقية < و <= و >= و > إلى ترتيب الرموز داخل كل سلسلة ، كما هو معرف بشفرة الرموز الخاصة المستخدمة .

مثال (٩-٢٤)

أعتبر توضيح متغير السلسلة التالي :

```
TYPE colors = PACKED ARRAY [1..5] OF char;  
VAR c1,c2,c3 : colors;
```

افترض الآن أن متغيرات السلسلة محدد لها قيم السلسلة التالية .

```
c1 := 'black';  
c2 := 'white';  
c3 := 'green';
```

فيما يلي عدة تعبيرات بوليان تشمل هذه المتغيرات ، ومعها القيم المناظرة لكل تعبير .

القيمة	تعبير بوليان
صحيح	( لأن 'b' < 'w' ) c1 < c2
صحيح	( لأن 'g' < 'w' ) c3 < c2
خطأ	( لأن 'l' < 'r' ) c1 > 'brown'
صحيح	( لأن 'o' > 'e' ) 'gross' > c3

ويجب أن يكون مفهوما أن استخدام كل محتوى المنظومة كعنصر يمكن التأثير عليه في تعبير بولياني يكون صحيحا إذا ما كانت المنظومة تمثل متغير سلسلة فقط . مثل هذه المقارنات تسمح بأنواع أخرى من المنظومات .

وهناك سمة أخرى تصاحب متغيرات السلسلة ، وتشمل استخدام عبارات write , writeln . يمكن أن توجد متغيرات السلسلة داخل هذه العبارات ، متسببة على ذلك في كتابة كل محتوى السلسلة ( أى كل عناصر السلسلة المضغوطة ) ، دون الحاجة إلى مكونات الدورة .



## مثال (٩-٢٥)

فيما يلي جزءا من برنامج بسكال :

```
.
.
VAR factory : PACKED ARRAY [1..11] OF char;
BEGIN
.
.
factory := 'Seattle, WA';
.
.
writeln(' Factory Location: ',factory);
.
.
END.
```

لاحظ أن عبارة writeln تحتوي على ثابت سلسلة ومتغير سلسلة .

عند تنفيذ هذا البرنامج نحصل على المخرجات التالية .

Factory Location: Seattle, WA

ويجب أن يكون مفهوما أن متغير السلسلة هو نوع المنظومة الوحيد الذي يمكن أن يوجد في عبارة writeln , write بهذه الطريقة . والأكثر من هذا فالإمكانية الشبيهة لذلك غير ممكنة عند استخدام عبارة read , readln , وذلك حتى مع متغيرات السلسلة . وعلى هذا ... فيجب إدخال السلاسل داخل الكمبيوتر رمزا رمزا باستخدام أحد أنواع مكونات الدورة .

## مثال (٩-٢٦)

فيما يلي جزءا من برنامج بسكال .

```
.
.
VAR factory : PACKED ARRAY [1..11] OF char;
i : 1..11;
BEGIN
.
.
FOR i := 1 TO 11 DO read(factory[i]);
writeln;
.
.
writeln(' Factory Location: ',factory);
.
.
END.
```

لاحظ الفرق بين عبارات المدخلات والمخرجات في هذا المثال . لاحظ أيضا الاختلافات بين هذا المثال والمثال السابق .

المقدرة على تعريف ومعاملة متغيرات السلسلة تفتح الباب لفئة واسعة من تطبيقات الكمبيوتر . والمثال التالي يعطى بعد التوضيح لذلك .

## مثال (٩-٢٧)

منتج حروف لعبة *Pig Latin* : هي صيغة شفرة من الإنجليزية ، وعادة ما يستخدمها الأطفال كأحدى ألعابهم . وتتكون كلمة latin من كلمة إنجليزية بنقل أول صوت ( عادة مايكون أول حرف ) إلى نهاية الكلمة ، ثم يضاف الحرف "a" وعلى هذا ... فكلمة cat تصبح "atca" ، وكلمة pascal تصبح "ascalPa" ، وكلمة pig latin تصبح "igpa atinla" وهكذا .

دعنا نكتب برنامج بسكال يقبل سطرا من نص مكتوب باللغة الإنجليزية ، ثم يطبع بعد ذلك السطر المناظر في pig latin . نفترض أن كل رسالة نصية يمكن كتابتها على سطر به 80 خانة ، مع وجود فراغ واحد بين الكلمات المتتالية على السطر . ( نحتاج في واقع الأمر أن لاتزيد رسالة pig latin عن 80 خانة . وعلى هذا ... فيجب أن تكون الرسالة الأصلية أقل من 80 خانة بعض الشيء ، حيث إن رسالة pig latin تكون أطول بإضافة حرف a لكل كلمة من كلمات السطر ) . وللتبسيط ننقل أول حرف فقط ( وليس أول صوت ) من كل كلمة . كما أننا نهمل أيضا أى اعتبارات خاصة يمكن أن تعطى للحروف الكبيرة وعلامات التنقيط .

وتحتوى عملية الحسابات الشاملة الخطوات الرئيسية التالية :

(١) وضع قيم ابتدائية لكل المنظومات ( متغيرات السلسلة ) ، وذلك بتحديد فراغات لكل العناصر .

(٢) قراءة المحتوى الكلى للسطر داخل الكمبيوتر ( عدة كلمات ) .

(٣) تحديد عدد الكلمات الموجودة في السطر ( وذلك بعد عدد الفراغات الفردية التى يتبعها حرف ) .

(٤) إعادة ترتيب الكلمات في صورة pig latin ، وذلك كلمة كلمة كما يلى :

(أ) تحديد نهاية كل كلمة .

(ب) نقل أول حرف إلى نهاية الكلمة .

(ج) إضافة حرف "a" إلى نهاية كل كلمة معدلة .

(هـ) كتابة كل محتويات السطر الجديد .

ونستمر في إعادة هذا الإجراء حتى يقرأ الكمبيوتر سطرا من نص ، تكون أول كلمة ( أو أول ثلاثة أحرف الأولى فيه ) هى end .

ولتنفيذ هذا العمل ، فإننا نستخدم مشيرين "pointers" اسمهما p1 ، p2 على التوالى . يحدد أول مشير p1 موقع بداية كلمة معينة داخل السطر الأسمى من النص . ويحدد المشير الثانى p2 نهاية الكلمة . لاحظ أن الرمز الموجود في العمود الذى يسبق رقم العمود p1 يكون فراغا ( فيما عدا أول كلمة ) . كما أن الرمز الموجود في العمود الذى يلى العمود رقم p2 يكون فراغا أيضا .

دعنا نستخدم طريقة الأجزاء مرة أخرى في إعداد هذا البرنامج . قبل مناقشة كل جزء من الأجزاء ، فإننا نعرف متغيرات البرنامج التالية :

- english = متغير سلسلة ( أى منظومة مضغوطة حرفية ) يمثل السطر الأصلي من النص .
- latin = متغير سلسلة يمثل السطر الجديد من النص ( أى سطر pig latin ) .
- flag = متغير بوليان يظل صحيحا true حتى يتم إدخال كلمة "end" ، وذلك ليسمح بتكرار تنفيذ البرنامج .
- words = متغير من النوع الصحيح يحدد عدد الكلمات الموجودة في السطر المعطى من النص .
- n = متغير من النوع الصحيح يستخدم كعدد للكلمات ( words, ... , 2 , 1 , = n ) .
- count = متغير من النوع الصحيح يستخدم كعدد الرموز داخل كل سطر من الأسطر ( count = 1, 2, ... , 80 ) .
- وهذه المتغيرات بالإضافة بالطبع إلى المتغيرين الصحيحين p1 , p2 الذين سبق ذكرهما .

دعنا نعود الآن إلى التخطيط الشامل للبرنامج الذى سبق ذكره . يمكن أداء أول خطوة ( وهى وضع قيم ابتدائية للمنظومات ) بطريقة مباشرة طبقا للإجراء التالى .

```
PROCEDURE initialize;
(* initialize the arrays with blank spaces *)
BEGIN
  FOR count := 1 TO 80 DO
    english [count] := ' ';
    latin := english
  END;
```

ويمكن أداء الخطوة الثانية بإجراء بسيط أيضا . ويحتوى هذا الإجراء على دورة شرطية ( مكون WHILE-DO ) يستمر فى قراءة الرموز من النهاية الطرفية حتى تكتشف نهاية السطر ( أى حتى تصبح الدالة eoln صحيحة true ) . هذا التسلسل الرموز يصبح عناصر المتغير السلسلة english . وفيما يلى المحتويات الكلية للإجراء .

```
PROCEDURE readinput;
(* read one line of English *)
BEGIN
  count := 0;
  WHILE NOT eoln DO
    BEGIN
      count := count + 1;
      read(english [count])
    END;
  readln
END;
```

الخطوة 3 من التخطيط الشامل للبرنامج مباشرة أيضا . ويمكننا أن نفحص ببساطة السطر لمعرفة ما إذا كان هناك خانة فارغة يتبعها حرف . عند ذلك تزداد الكلمة counter ( عداد words ) فى كل مرة يظهر فيها خانة فارغة . وفيما يلى جزء عد الكلمات :

```
PROCEDURE countwords;
(* scan the line and count the number of words *)
BEGIN
  words := 1;
  FOR count := 1 TO 79 DO
    IF (english [count] = ' ') AND (english [count + 1] <> ' ')
      THEN words := words + 1
  END;
```

دعنا نعتبر الآن الخطوتين رقم 4 ورقم 5 من التخطيط الشامل للبرنامج . وهذه الخطوات هي في واقع الأمر قلب البرنامج . ويمكن دمجهما في واقع الأمر في إجراء واحد ، حيث إن الخطوة رقم 5 لا تتطلب إلا عبارة فردية واحدة فقط . ومن ناحية أخرى ... فإن الخطوة رقم 4 تتطلب ثلاث عمليات منفصلة ، بالرغم من ارتباطها مع بعضها .

ويجب أن نعرف أولا نهاية كل كلمة ، وذلك بإيجاد أول خانة فارغة بعد  $p1$  . عند ذلك نحدد الرموز التي تتكون منها الكلمة في pig latin ، وذلك بأخر حرف في نهاية الكلمة . ويجب أن يؤدي ذلك بحرص ، حيث إن السطر الجديد من النص يكون أطول من السطر الأصلي ( وذلك بسبب حروف "a" الإضافية ) . وعلى هذا ... فإن الحروف في أول كلمة pig latin تمثل المواقع من  $p1$  وحتى  $p2+1$  . والحروف في ثاني كلمة تمثل المواقع من  $p1+1$  إلى  $p2+1$  ( لاحظ أن هذه هي قيم جديدة لكل من  $p2$  ,  $p1$  ) وهكذا . ويمكن تعميم هذه القواعد كما يلي :

**أولاً** ، تنتقل كل حروف الكلمة رقم  $n$  ، فيما عدا الحرف الأول من السطر الأصلي إلى السطر الجديد . ويمكن تحقيق ذلك بكتابة .

```
FOR count := p1 TO p2 - 1 DO
  latin [count + (n - 1)] := english [count + 1];
```

اعتبر العبارة التالية :

```
latin [count + (n - 1)] := english [count + 1];
```

داخل أي كلمة تتسبب هذه العبارة في أن ثاني حرف إنجليزي يصبح أول حرف pig latin ، وثالث حرف إنجليزي يصبح ثاني حرف pig latin وهكذا . وتستخدم الكمية (  $n-1$  ) في تضبيط موقع كل كلمة pig latin بسبب الحروف الزائدة التي يتم إضافتها ( أي حروف a في نهاية كل كلمة ) وعلى هذا ... فإن الكلمة الثانية (  $n=1$  ) تضبط ( أي ترحل إلى اليمين ) بإزاحتها خانة واحدة ، وذلك بسبب وجود a في نهاية أول كلمة والكلمة الثالثة (  $n=3$  ) تضبط بإزاحتها خانتين فقط .

ويعد تنفيذ الدورة المذكورة أعلاه لكل الحروف الموجودة داخل الكلمة باستثناء الحرف الأول ، فإننا نضيف أول حرف يتبعه الحرف 'a' . ويتم تحقيق ذلك بالطريقة التالية :

```
latin [p2 + (n-1)] := english [p1];
latin [p2 + n] := 'a';
```

ثم نعيد وضع قيمة  $p1$  إلى :

```
p1 := p2 + 2
```

وذلك للإعداد للكلمة التالية .

وتعاد هذه المجموعة من العمليات لكل كلمة من كلمات السطر الأصلي . وبعد ذلك يكتب السطر الجديد ، والذي يحتوي على المكافئ من pig latin للسطر الأصلي .

وفيما يلي إجراء تحقيق ذلك .

```

PROCEDURE rearrangewords;
(* rearrange each word into piglatin, then write out the entire line *)
BEGIN
  p1 := 1;
  FOR n := 1 TO words DO
    BEGIN
      (* locate end of word *)
      count := p1;
      WHILE english [count] <> ' ' DO count := count + 1;
      p2 := count - 1;

      (* transpose first letter and add 'a' *)
      FOR count := p1 TO p2 - 1 DO
        latin [count + (n - 1)] := english [count + 1];
      latin [p2 + (n - 1)] := english [p1];
      latin [p2 + n] := 'a';
      p1 := p2 + 2
    END;
    writeln(latin)
  END;
END;
```

دعنا نعتبر الآن الجزء الرئيسى . هذا الجزء ، البرنامج لايزيد عن أن يكون عبارة عن رسالة ابتدائية يتبعها مكون REPEAT-UNTIL يسمح بتكرار تنفيذ البرنامج ( حتى تظهر كلمة end كأول ثلاثة أحرف فى النص الإنجليزى ) . ونبدأ بصفة خاصة بتحديد قيمة true لـ flag ، ثم نغير هذه القيمة إلى false عند ظهور كلمة "end" داخل الدورة . ويمكن أن يوضع مكون IF-THEN داخل الدورة لاختبار هذا الشرط . وعند ذلك يفصل إجراء الدورة بمجرد أن تصبح قيمة flag خاطئة false .

وفيما يلي محتويات البرنامج . لاحظ إضافة العديد من التعليقات إلى الإجراءات التى سبق ذكرها .

```

PROGRAM piglatin(input,output);
(* THIS PROGRAM ACCEPTS A LINE OF ENGLISH TEXT AND CONVERTS IT TO PIGLATIN. *)
(* Word pointers: p1 -> beginning of word
                  p2 -> end of word *)
TYPE line = PACKED ARRAY [1..80] OF char;
VAR english,latin : line;
    p1,p2,count,n,words : integer;
    flag : boolean;

PROCEDURE initialize;
(* initialize the arrays with blank spaces *)
BEGIN
  FOR count := 1 TO 80 DO
    english [count] := ' ';
    latin := english
  END; (* initialize *)

PROCEDURE readinput;
(* read one line of English *)
BEGIN
  count := 0;
  WHILE NOT eoln DO
    BEGIN
      count := count + 1;
      read(english [count])
    END;
  END;
```

(تكملة البرنامج فى الصفحة التالية)

```

    readln
END; (* readinput *)

PROCEDURE countwords;
(* scan the line and count the number of words *)
BEGIN
    words := 1;
    FOR count := 1 TO 79 DO
        IF (english [count] = ' ') AND (english [count + 1] <> ' ')
            THEN words := words + 1
    END; (* countwords *)

PROCEDURE rearrangewords;
(* rearrange each word into piglatin, then write out the entire line *)
BEGIN
    p1 := 1;
    FOR n := 1 TO words DO
        BEGIN
            (* locate end of word *)
            count := p1;
            WHILE english [count] <> ' ' DO count := count + 1;
            p2 := count - 1;

            (* transpose first letter and add 'a' *)
            FOR count := p1 TO p2 - 1 DO
                latin [count + (n - 1)] := english [count + 1];
            latin [p2 + (n - 1)] := english [p1];
            latin [p2 + n] := 'a';
            p1 := p2 + 2
        END;
        writeln(latin)
    END; (* rearrangewords *)

BEGIN (* main action block *)
    writeln(' Piglatin generator');
    writeln;
    writeln(' Type ''end'' when finished');
    writeln;
    flag := true;
    REPEAT (* process a line of text *)
        initialize;
        readinput;
        IF (english [1] = 'e') AND (english [2] = 'n') AND
            (english [3] = 'd') THEN flag := false;
        countwords;
        rearrangewords
    UNTIL flag = false
END.

```

لاحظ أن هذا البرنامج يحتوى على تحديد محتويات إحدى المنظومات لمنظومة أخرى ( فى إجراء initialize ) وتحديد عناصر فردية لمنظومة ( فى readinput , rearrangewords ) . لاحظ أيضا أن محتوى متغير سلسلة ( أى محتوى سلسلة مضغوطة ) يكتب بعبارة writeln واحدة فى الإجراء rearrangewords . ويجب على عملية الإدخال المناظرة بالطبع أن تؤدي عنصرا عنصرا كما فى الإجراء readinput .

دعنا نعتبر الآن ماذا يحدث عندما ينفذ البرنامج . فيما يلي حواراً تمطياً ، موضوع خط فيه تحت استجابة المستفيد .

```
Piglatin generator
Type 'end' when finished
Pascal is a popular structured programming language
ascalPa sia aa opularpa tructuredsa rogrammingpa anguagela
baseball is the great American pastime,
aseballba sia heta reatga mericanAa astime,pa

though there are many who prefer football
houghta hereta reaa anyma howa referpa ootballfa

please do not sneeze in the computer room
leasepa oda otna neezesa nia heta omputerca oomra

end
ndea
```

لاحظ أن البرنامج لا يحتوى على أى إشارة فى النص لعلامات تنقيط ، أو حروف كبيرة ، أو أصوات مزدوجة الحرف ( مثل th أو sh وخلافه ) . وهذه الأمور متروكة ليجربها القارئ كتمارين .

## ٦ - مؤشرات منظومة متغيرة الطول : VARIABLE-LENGTH ARRAY PARAMETERS

لقد رأينا بالفعل أنه يمكن تمرير كل محتويات المنظومة إلى إجراء ، أو إلى دالة ، على أن تكون المؤشرات الفعلية والمؤشرات الرسمية المناظرة لها من نفس النوع الصريح ، وتحتوى على ذلك على نفس عدد العناصر ( انظر قسم ٣ من هذا الفصل ويصفه خاصة مثال ٩ - ١٦ ) . وفى بعض التطبيقات قد يكون مطلوباً وجود ( داخل إجراء أو دالة ) مؤشر رسمى لا يكون محدد له حد أقصى لعدد عناصره بصورة مطلقة . وهذه السمة تسمح بالاتصال بالإجراء أو بالدالة من عدة مواقع بمؤشرات فعلية مختلفة الحجم عند كل نقطة من نقاط الاتصال .

يمكن استخدام المؤشرات الرسمية ذات الطول المتغير للمنظومة داخل إجراء أو دالة ، على أن توضح هذه المؤشرات بطريقة خاصة . وتعرف مثل هذه المؤشرات بأنها مؤشرات فعلية لمنظومة متطابقة .

ويوسع استخدام المؤشرات المتطابقة لمنظومة من تعميم برنامج البسكال . ومن ناحية أخرى ... فإن المتطلبات التكوينية المصاحبة لاستخدامها معقدة . وقد يختار المبرمجون المبتدئون على ذلك تجنب استخدامها حتى يصبحوا أكثر اعتياداً على صيغ مؤشرات المرور الأخرى .

دعنا نعتبر أولاً مؤشرات منظومة متطابقة ذات بعد واحد . وطريقة تعريفها تعتمد على ما إذا كانت هذه المؤشرات هى مؤشرات قيمة ، أو مؤشرات متغير ، كما يتضح ذلك من المثال التالى .

مثال (٩-٢٨)

افرض أن sample هو إجراء يستخدم مؤشراً متغيراً رسمياً لمنظومة تسمى list من النوع الحقيقى . ويمكن على ذلك كتابة عنوان الإجراء كما يلى :

```
PROCEDURE sample (VAR list : ARRAY [first..last : integer] OF real);
```

وعلى هذا فإننا نرى أن list عبارة عن منظومة حقيقية النوع ، لها بعد واحد ، وقيم فهرسها هي أرقام صحيحة تتراوح من first إلى last . وتتحدد القيم العددية لكل من first , last بواسطة المؤشر الفعلي المناظر ، والذي يجب أن يكون منظومة حقيقية النوع ذات بعد واحد ، لها فهرس صحيح النوع .

اعتبر على سبيل المثال منظومة من النوع الحقيقي ، لها بعد واحد ، اسمها item ، ويظهر توضيحها في المجموعة الرئيسية للبرنامج ، أي :

```
VAR item : ARRAY [1..100] OF real;
```

افرض الآن أن item هو مؤشر فعلى يمر إلى sample ، أي :

```
sample(item);
```

وعلى هذا ... يأخذ first القيمة 1 ، ويأخذ last القيمة 100 ، بحيث تعامل list كأنها منظومة بها 100 عنصر من النوع الحقيقي أثناء الاتصال بالإجراء .

فإذا ما عدلت عناصر list داخل الإجراء ، فإن عناصر المصفوفة المعدلة تعود إلى item كما هو الحال دائما مع متغير المؤشر الرسمي .

مثال (٩-٢٩)

فيما يلي تخطيط هيكلية لبرنامج بسكال ، يستخدم دالة تقوم بعدد الكلمات الموجودة في السطر . ويهدف التوضيح ... دعنا نستخدم مؤشر قيمة متوافقة للمنظومة داخل الدالة .

```
PROGRAM dummy(input,output);
TYPE length = 1..80;
    words = 1..20;
VAR heading : PACKED ARRAY [1..12] OF char;
    message : PACKED ARRAY [1..80] OF char;
    n1,n2 : words;

FUNCTION count(text : PACKED ARRAY [start..stop : length] OF char) : words;
BEGIN
    .
    .
    .
    count := . . .
END;

BEGIN (* main action statements *)
    .
    .
    n1 := count(heading);
    .
    .
    n2 := count(message);
    .
    .
END.
```



يحتوى المثال على معالم متعددة تتطلب بعض التوضيح . لاحظ أولا text تعتبر منظومة مضغوطة ، بها 12 عنصرا ، وذلك أثناء الإشارة الأولى إلى الدالة ، كما أن text تعتبر منظومة مضغوطة بها 80 عنصرا ، وذلك أثناء الإشارة الثانية إلى الدالة . وتعود الدالة في كل حالة برقم صحيح ، تقع قيمته بين 1 ، 20 . ( وهذا بافتراض أن كل سطر من أسطر النص يحتوى على كلمة واحدة على الأقل ، ولا يحتوى على أكثر من 20 كلمة ) .

يلى ذلك ان المنظومتين المضغوطتين message , heading ، والمستخدمتين كمؤشرين فعليين في الإشارة إلى الدالة هما متغيران سلسلة صحيحان . ( نفترض أن السلسلة ممثلة بواسطة هذين المتغيرين تقرأ داخل الكمبيوتر ، وإلا فتحدد قيمها قبل الإشارة إلى الدالة ) . وعلى أية حال ... فإن مؤشر القيمة الرسمى text ( أى مؤشر قيمة متوافقة للمنظومة ) لا يعتبر متغير سلسلة . والخواص الخاصة بمتغيرات السلسلة التى سبق مناقشتها فى قسم ه من هذا الفصل لا يمكن استخدامها داخل الدالة . وهذا القيد يسرى على المنظومات المضغوطة من النوع الحرفى المستخدمة كمؤشرات منظومة متوافقة .

يمكن أن يكون للمؤشرات منظومة متوافقة أبعاد متعددة أو بعد واحد . وبصفة عامة ... تطبق نفس القاعدة ، بالرغم من وجود قيد واحد خاص باستخدام الضغط . وبالتحديد فإن البعد الداخلى جدا ( أى آخر بعد من الداخل ) يمكن ضغطه . فإذا ما استخدم الضغط بهذه الطريقة ، فيجب تعريف مؤشر المنظومة المتوافقة بأنه منظومة من المنظومات المضغوطة ، كما هو موضح أدناه .

#### مثال (٩-٣٠)

اعتبر عنوان الإجراء التالى ، الذى يستخدم مؤشر منظومة متوافقة ذات بعدين :

```
PROCEDURE sample (text : ARRAY [first..last : integer]
OF PACKED ARRAY [c1..cn : integer] OF char);
```

لاحظ أن text هو مؤشر قيمة منظومة متوافقة ، وعناصره هي حروف فردية .

إذا ما نظر على ذلك لصفحة من النص page of text ، حيث أول فهرس يمثل رقم السطر ، وثانى فهرس يمثل موقع الحرف داخل سطر معين . لاحظ أيضا أن كل من الفهرسين من النوع الصحيح ، بالرغم من أن عناصر المنظومة نفسها من النوع الحرفى .

وهناك قيود معينة يجب مراعاتها عند استخدام مؤشرات منظومة متوافقة . لقد ناقشنا بالفعل اثنين من هذه القيود ، وهما أنه لا يمكن معاملة منظومة مضغوطة من النوع الحرفى كمؤشر سلسلة داخل الإجراء أو الدالة التى تمر إليها ( أنظر مثال ٩-٢٩ ) . ويمكن استخدام الضغط مع البعد الداخلى جدا فقط بالنسبة للمنظومة متعددة الأبعاد . وهناك قيد إضافى خاص بمرور مؤشر منظومة متوافقة إلى إجراء آخر ، أو إلى دالة أخرى . فى مثل هذه الحالات يجب تعريف المؤشر الرسمى الموجود داخل الجزء الثانوى بأنه مؤشر متغير منظومة متوافقة ، بدلا من أنه مؤشر قيمة منظومة متوافقة .

مثال (٩-٣١)

فيما يلي تخطيطا هيكليا لبرنامج بسكال

```

PROGRAM nothing(input,output);
VAR item : ARRAY [1..100] OF real;

FUNCTION largest (VAR numbers : ARRAY [start..stop : integer] OF real) : real;
BEGIN
    .
    .
    largest := . . .
END;

PROCEDURE process (VAR list : ARRAY [first..last : integer] OF real);
VAR count : real;
BEGIN
    .
    .
    count := largest(list);
    .
    .
END;

BEGIN (* main block *)
    .
    .
    process(item);
    .
    .
END.

```

تمر في هذا المثال المنظومة الحقيقية التي بها 100 عنصرا ، والمسمى item إلى الإجراء process . والمؤشر الرسمي المناظر هو مؤشر متغير منظومة متوافقة يسمى list . وتمر هذه المنظومة من خلال process إلى الدالة largest كما أن المؤشر الرسمي المناظر في largest هو أيضا مؤشر متغير لمنظومة متوافقة يسمى numbers .

لاحظ أن مؤشر المنظومة المتوافقة الموجود في process ( وهو list ) يصبح فيما بعد مؤشرا فعليا عندما يشار إلى الدالة largest . وعلى هذا ... يجب تعريف numbers ( المؤشر الرسمي داخل largest ) كمؤشر متغير منظومة متوافقة ، بدلا من أنه مؤشر قيمة منظومة متوافقة .

وهناك قيد آخر يجب ملاحظته عند تعريف مؤشري منظومة متوافقة معا . في مثل هذه الحالة يجب أن تكون المؤشرات الفعلية المناظرة في أي إشارة فردية من نفس النوع . وهذا القيد موضح في المثال التالي .

مثال (٩-٣٢)

جمع جدولين من الأعداد ، دعنا نعتبر مرة أخرى برنامج جمع جدولين من الأعداد ، والذي سبق ذكره في مثال ٩ - ١١ . دعنا - على أية حال - نستخدم الآن مؤشرات منظومة متوافقة في تمثيل الجدولين داخل أجزاء البرنامج .

```

PROGRAM table3(input,output);

(* THIS PROGRAM READS IN TWO TABLES OF INTEGERS,
   CALCULATES THE SUMS OF THEIR RESPECTIVE ELEMENTS,
   AND WRITES OUT THE RESULTING TABLE CONTAINING THE SUMS *)

(* THE PROGRAM MAKES USE OF CONFORMANT ARRAY PARAMETERS *)

VAR a,b,c : ARRAY [1..20,1..30] OF integer;
    row,nrows : 1..20;
    col,ncols : 1..30;

PROCEDURE readinput (VAR t : ARRAY [firstrow..lastrow : integer;
                                   firstcol..lastcol : integer] OF integer);

(* Read the elements of one table *)

BEGIN
    FOR row := 1 TO nrows DO
        BEGIN
            writeln(' Enter data for row no. ',row:2);
            FOR col := 1 TO ncols DO read(t[row,col]);
            writeln
        END
    END;
END; (* readinput *)

PROCEDURE computesums (VAR t1,t2,t3 : ARRAY [firstrow..lastrow : integer;
                                             firstcol..lastcol : integer] OF integer)

(* Sum the elements of two similar tables *)

BEGIN
    FOR row := 1 TO nrows DO
        FOR col := 1 TO ncols DO
            t3[row,col] := t1[row,col] + t2[row,col]
        END;
    END; (* computesums *)

PROCEDURE writeoutput (t : ARRAY [firstrow..lastrow : integer;
                                   firstcol..lastcol : integer] OF integer);

(* Write out the elements of a table *)

BEGIN
    writeln(' Sums of the elements:');
    writeln;
    FOR row := 1 TO nrows DO
        BEGIN
            FOR col := 1 TO ncols DO write(t[row,col]:4);
            writeln
        END
    END;
END; (* writeoutput *)

(تكملة البرنامج في الصفحة التالية)

```

```
BEGIN (* main action block *)
  write(' How many rows? (1..20) ');
  readln(nrows);
  writeln;
  write(' How many columns? (1..30) ');
  readln(ncols);
  writeln;
  writeln(' First table:');
  writeln;
  readinput(a);
  writeln(' Second table:');
  writeln;
  readinput(b);
  computesums(a,b,c);
  writeoutput(c)
END.
```

ومن المفيد مقارنة صيغة هذا البرنامج من البرنامج الموجود في مثال ٩ - ١٦. لاحظ على سبيل المثال أن تعريف النوع غير موجود في الصيغة الحالية . بدلا من ذلك توجد مواصفات المنظومة داخل التوضيحات المتغيرات .

ومن المهم جدا معرفة حقيقة أن كل الإجراءات تستخدم مؤشرات منظومة متوافقة كمؤشرات رسمية . يستخدم إجراءان ، وهما readinput , computesums مؤشرات متغير منظومة متوافقة ، بينما يستخدم الإجراء الثالث وهو writeoutput مؤشر قيمة منظومة متوافقة . لاحظ أن الثلاثة مؤشرات منظومة متوافقة t1 , t2 , t3 معرفة معا في الإجراء computesums . لاحظ أيضا أن الإشارة المناظرة إلى computesums داخل المجموعة الرئيسية تحتوى على ثلاثة مؤشرات فعلية ، وهى a , b , c ، وكلها من نفس النوع كما هو مطلوب .

وقبل ترك هذا القسم ، فإننا نذكر القارئ مرة أخرى بأن استخدام مؤشرات منظومة متوافقة معقد بعض الشيء وقد لا يكون مناسباً على أية حال مع المبرمجين المبتدئين . وأكثر من ذلك ... فإن بعض صيغ البسكال لاتستخدم مؤشرات منظومة متوافقة . وهذا صحيح بصفة خاصة مع أجهزة الميكروكمبيوتر . وحتى إذا ما كانت هذه السمة متاحة ، فيجب أن يتم التدريب عليها جيداً قبل البدء فى استخدامها فى البرامج .

## Review Questions

## أسئلة للمراجعة :

- (١) ماهى الأربعة أنواع المختلفة للبيانات المرتبة ؟
- (٢) ماهى الخواص الخاصة بمنظومة كيانات مرتبة النوع ؟
- (٣) اقترح طريقة عملية لرؤية منظومة ذات بعد واحد .
- (٤) مامعنى عنصر المنظومة ؟ وما نوع البيانات التى يمكن استخدامها كعناصر فى منظومة ؟
- (٥) مامعنى فهرس ( أو دليل ) ؟ ما نوع البيانات التى يمكن استخدامها كفهرس ؟ ( قارن إجابتك مع إجابتك للسؤال السابق ) .
- (٦) كيف يمكن الاتصال بعنصر واحد من عناصر المنظومة ؟
- (٧) وضح كيف يمكن تعريف منظومة داخل جزء توضيح المتغيرات فى البرنامج .
- (٨) لخص قواعد استخدام عنصر منظومة داخل تعبير ، أو داخل عبارة تحديد ، أو داخل عبارة read أو write .. الخ . كيف يمكن كتابة الفهارس ؟ وماهى القيود الموضوعة على الطريقة التى تكتب بها الفهارس ؟

- (٩) هل يمكن استخدام عناصر البيانات المتعددة كعناصر منظومة ؟
- (١٠) وضح كيف يمكن استخدام منظومة في تمثيل سلسلة .
- (١١) اقترح طريقة عملية لرؤية منظومة ذات بعدين .
- (١٢) لخص قواعد تعريف منظومة ذات أبعاد متعددة كجزء من توضيح المتغيرات . هل يجب أن تكون كل الفهارس من نفس نوع البيانات ؟
- (١٣) لخص قواعد الإشارة إلى عنصر من عناصر منظومة متعددة الأبعاد .
- (١٤) وضح سبب الحاجة إلى دورات متداخلة لبعض تطبيقات المنظومات متعددة الأبعاد . ماهو الفرض من كل دورة ؟
- (١٥) صف كيف يمكن التعبير عن منظومة ذات بعدين بأنها منظومة ذات بعد واحد ، وكل عنصر من عناصرها الفردية عبارة عن منظومة ذات بعد واحد . هل يمكن توسيع هذا المفهوم ليشمل منظومات لها أبعاد أكثر من ذلك ؟
- (١٦) صف كيف يمكن توضيح عدة منظومات من نفس النوع عن طريق تعريف نوع المنظومة ، ثم توضيح أن المنظومات الفردية عبارة عن متغيرات من نفس النوع . هل يجب على كل المنظومات الموضحة بهذه الطريقة أن يكون لها نفس المدى ؟
- (١٧) كيف يمكن تحديد جميع عناصر منظومة لمنظومة أخرى باستخدام عبارة تحديد واحدة ؟ ماهى القيود الموضوعة على هاتين المنظومتين ؟
- (١٨) هل يمكن أن تظهر جميع محتويات المنظومة في تعبير عددي أو تعبير بولياني ؟
- (١٩) هل يمكن قراءة جميع محتويات المنظومة داخل الكمبيوتر باستخدام عبارة read أو readln واحدة ؟
- (٢٠) هل يمكن كتابة جميع محتويات المنظومة خارج الكمبيوتر باستخدام عبارة write أو writeln واحدة ؟
- (٢١) ماهى القيود الموضوعة على مرور جميع محتويات المنظومة إلى أجزاء أو إلى دالة ؟
- (٢٢) صف طريقة مقننة لمرور جميع محتويات منظومة إلى إجراء أو إلى دالة .
- (٢٣) هل يمكن مرور عنصر منظومة واحد إلى إجراء أو إلى دالة ؟
- (٢٤) ماهى المنظومة المضغوطة ؟ وكيف تختلف المنظومات المضغوطة عن المنظومات المعتادة ؟
- (٢٥) ماهى المميزات والعيوب الممكنة التي تصاحب استخدام المنظومات المضغوطة ؟
- (٢٦) لأي نوع من أنواع التطبيقات تكون المنظومات المضغوطة ذات ميزة خاصة ؟
- (٢٧) هل يمكن للمنظومة المضغوطة أن يفك ضغطها ؟ وهل يمكن للمنظومة غير المضغوطة أن تضغط ؟ وضح ذلك .

- (٢٨) لخص القواعد المصاحبة لاستخدام الإجرائين القياسيين pack , unpack .
- (٢٩) هل يمكن تمرير جميع محتويات منظومة مضغوطة إلى إجراء أو إلى دالة ؟
- (٣٠) هل يمكن تمرير عنصر واحد من عناصر منظومة مضغوطة إلى إجراء أو إلى دالة ؟
- (٣١) ماهو متغير السلسلة ؟ وماهى العلاقة بين المنظومات المضغوطة ومتغيرات السلاسل ؟
- (٣٢) صف العمليات الخاصة بالمنظومة المضغوطة التى يمكن استخدامها مع سلاسل أو مع متغيرات سلاسل فقط .
- (٣٣) كيف تفسر المؤثرات العلاقية عند استخدامها فى وصل سلاسل أو متغيرات سلاسل لتكوين تعبير بوليان ؟ وبصفة خاصة كيف تفسر المؤثرات < و <= و <> و >= ؟
- (٣٤) ماهو مؤشر المنظومة المتوافقة ؟ وفى أى غرض يستخدم ؟
- (٣٥) لخص قواعد تعريف مؤشرات منظومة متوافقة ذات بعد واحد . ميز بين مؤشرات قيمة منظومة متوافقة ، ومؤشرات متغير منظومة متوافقة .
- (٣٦) لخص قواعد تعريف مؤشر منظومة متوافقة ذات أبعاد متعددة . قارن ذلك بالقواعد المذكورة فى إجابة السؤال السابق .
- (٣٧) هل يمكن استخدام الضغط مع منظومة متعددة الأبعاد ؟ وضح ذلك بالتفصيل .
- (٣٨) افرض أن إحدى المنظومات المضغوطة من النوع الحرفى تم تمريرها إلى إجراء أو إلى دالة كمؤشر منظومة متوافقة . هل يمكن معاملة هذه المنظومة كمتغير سلسلة داخل الإجراء أو داخل الدالة ؟
- (٣٩) افرض أن مؤشر رسمى لمنظومة متوافقة داخل إجراء معين سوف يمرر إلى إجراء آخر . ( أى أن المؤشر الرسمى لمنظومة متوافقة سيصبح مؤشرا فعليا ليمر إلى إجراء ثانوى ) . باى طريقة يجب تعريف مؤشر منظومة متوافقة داخل الإجراء الثانوى ؟
- (٤٠) افرض أنه تم تعريف اثنين أو أكثر من مؤشرات رسمية لمنظومة متوافقة معا داخل عنوان أحد الإجراءات . ماهو الشرط الذى يجب تحقيقه بالمؤشرات الفعلية لمنظومة متوافقة تناظر هذه المؤشرات ؟
- (٤١) هل مؤشرات المنظومة المتوافقة موجودة فى كل صيغ البسكال ؟

## Solved Problems

## مسائل محلولة :

(٤٢) فيما يلي عدة توضيحات صحيحة لمنظومات :

- (a) VAR values : ARRAY [0..100] OF real;
- (b) VAR list : ARRAY [-10..10] OF real;
- (c) TYPE days = (sun,mon,tues,wed,thurs,fri,sat);  
VAR calories : ARRAY [days] OF integer;
- (d) TYPE days = (sun,mon,tues,wed,thurs,fri,sat);  
VAR sunshine : ARRAY [1..365] OF days;
- (e) TYPE color = (blue,brown,gray,green,black);  
size = (small,medium,large);  
VAR coats : ARRAY [size] OF color;
- (f) TYPE color = (blue,brown,gray,green,black);  
size = (small,medium,large);  
VAR coats : ARRAY [size,color] OF integer;
- (g) TYPE color = (blue,brown,gray,green,black);  
size = (small,medium,large);  
VAR coats : ARRAY [size] OF ARRAY [color] OF integer;
- (h) VAR parts : ARRAY ['A'..'Z', 1..10000] OF integer;
- (i) CONST max1 = 2000;  
max2 = 5000;  
TYPE range1 = 1..max1;  
range2 = 1..max2;  
VAR tally : ARRAY [range1,range2] OF char;
- (j) VAR symbol : ARRAY [1..12, 1..20, 0..5] OF boolean;
- (k) TYPE days = (sun,mon,tues,wed,thurs,fri,sat);  
VAR sunshine : PACKED ARRAY [1..365] OF days;
- (l) VAR name : PACKED ARRAY [1..80] OF char;
- (m) VAR page : ARRAY [1..66] OF PACKED ARRAY [1..80] OF char;

(٤٣) التخطيطات الهيكلية التالية توضح استخدام منظومات وعناصر منظومات في مواقف برمجة .

- (a) PROGRAM sample(input,output);  
VAR square : ARRAY [1..100] OF integer;  
count : 1..100;  
BEGIN  
.  
.  
FOR count := 1 TO 100 DO square [count] := sqr(count);  
.  
.  
END.
- (b) PROGRAM sample(input,output);  
VAR table : ARRAY [1..20, 1..8] OF real;  
i,m : 1..20;  
j,n : 1..8;  
BEGIN

```

.
.
readln(m,n);
FOR i := 1 TO m DO
  BEGIN
    FOR j := 1 TO n DO read( table [i,j]);
    writeln
  END;
.
.
FOR i := 1 TO m DO
  BEGIN
    FOR j := 1 TO n DO write( table [i,j]);
    writeln
  END
END
(c) PROGRAM sample(input,output);
VAR value : ARRAY [0..100] OF real;
    index,max : 0..100;
BEGIN
  .
  .
  readln(max);
  .
  .
  value[0] := 0;
  FOR index := max DOWNTO 0 DO
    BEGIN
      value[index] := index MOD 8;
      IF value[index] < 5.0 THEN value[0] := value[0] + value[index]
    END;
  .
  .
  END.

(d) PROGRAM sample(input,output);
TYPE color = (red,white,blue,yellow,green);
VAR foreground : PACKED ARRAY [1..5] OF color;
    background : ARRAY [1..5] OF color;
    index : 1..5;
BEGIN
  .
  .
  FOR index := 1 TO 5 DO background[index] := . . .;
  .
  .
  pack(background,1,foreground);
  .
  .
  END.

(e) PROGRAM sample(input,output);
VAR city,office : PACKED ARRAY [1..3] OF char;
    order : 1..maxint;
BEGIN

```



```

.
.
city := 'CHI';
.
.
readln(order);
IF order < 100 THEN office := city
    ELSE IF order < 1000 THEN office := 'NYC'
        ELSE office := 'ATL';
.
.
writeln(office);
.
.
END.

```

لاحظ أن كلا من city , office هو متغير سلسلة .

(f) PROGRAM sample(input,output);

```

VAR list1 : ARRAY [1..40] OF integer;
    list2 : ARRAY [1..80] OF integer;

PROCEDURE sort(VAR items : ARRAY [first..last : integer] OF integer);
VAR . . . ; (* local variables *)
BEGIN
    .
    .
    (* process the elements of items *)
    .
    .
END;

BEGIN (* main action statements *)
    .
    .
    (* read elements of list1 *)
    .
    .
    sort(list1);
    .
    .
    (* read elements of list2 *)
    .
    .
    sort(list2);
    .
    .
END.

```

لاحظ أن كلا من list1 , list2 منظومة مختلفة في حجمها .

## Supplementary Problems

## مشاكل متكاملة :

(٤٤) التخطيط الهيكلي التالي يوضح مواقع مختلفة عديدة تشمل استخدام مصفوفات وعناصر مصفوفات ، بعضها مكتوب بطريقة خاطئة . عرف كل الأخطاء .

- (a) TYPE notes = (do,re,mi,fa,sol,la,ti);  
VAR tune : ARRAY [1..256] OF notes;
- (b) TYPE notes = (do,re,mi,fa,sol,la,ti);  
VAR count : ARRAY [notes] OF integer;
- (c) TYPE notes = (do,re,mi,fa,sol,la,ti);  
TYPE chorus = PACKED ARRAY [1..256] OF notes;  
VAR first,second,trio,coda : chorus;
- (d) VAR demo : ARRAY [12..-5, 0..48] OF real;
- (e) PROGRAM sample(input,output);  
VAR list : ARRAY [1..100] OF real;  
BEGIN  
.  
.  
list [2.0] := 5.5;  
.  
.  
END.
- (f) PROGRAM sample(input,output);  
VAR list : ARRAY [1..100] OF real;  
index : 1..100;  
BEGIN  
.  
.  
FOR index := -100 TO 100 DO  
IF index < 0 THEN list [index] := 0  
ELSE list [index] := index;  
.  
.  
END.
- (g) PROGRAM sample(input,output);  
VAR list : ARRAY [1..100] OF real;  
index : 1..100;  
max : real;  
BEGIN  
.  
.  
max := 0;  
FOR index := 1 TO 100 DO  
IF list [index] > max THEN max := list [index];  
.  
.  
END.

- (h) `PROGRAM sample(input,output);`  
`VAR item : ARRAY ['A'..'Z', 1..50] OF integer;`  
`BEGIN`  
`.`  
`.`  
`item [7] := 283;`  
`.`  
`.`  
`END.`
- (i) `PROGRAM sample(input,output);`  
`VAR size : ARRAY [1..50] OF integer;`  
`BEGIN`  
`.`  
`.`  
`size [12] := 41.5;`  
`.`  
`.`  
`END.`
- (j) `PROGRAM sample(input,output);`  
`VAR item : ARRAY ['A'..'Z', 1..50] OF integer;`  
`BEGIN`  
`.`  
`.`  
`item [7,'M'] := 283;`  
`.`  
`.`  
`END.`
- (k) `PROGRAM sample(input,output);`  
`VAR item : ARRAY ['A'..'Z', 1..50] OF integer;`  
`BEGIN`  
`.`  
`.`  
`item ['C',9] := 60;`  
`.`  
`.`  
`END.`
- (l) `PROGRAM sample(input,output);`  
`TYPE list = ARRAY [1..200] OF real;`  
`VAR sales, costs, profits : list;`  
`BEGIN`  
`.`  
`.`  
`profits := sales - costs;`  
`.`  
`.`  
`END.`

- (m) `PROGRAM sample(input,output);`  
`VAR name : PACKED ARRAY [1..5] OF char;`  
`BEGIN`  
`.`  
`.`  
`IF name = 'Jones' THEN writeln(name);`  
`.`  
`.`  
`END.`
- (n) `PROGRAM sample(input,output);`  
`VAR table : ARRAY [1..20, 1..80] OF integer;`  
`BEGIN`  
`.`  
`.`  
`read(table);`  
`.`  
`.`  
`write(table);`  
`.`  
`.`  
`END.`
- (o) `PROGRAM sample(input,output);`  
`VAR table : ARRAY [1..1000, 1..66, 1..132] OF integer;`  
`i,j,k : integer;`  
`BEGIN`  
`.`  
`.`  
`FOR i := 1 TO 1000 DO`  
`FOR j := 1 TO 66 DO`  
`FOR k := 1 TO 132 DO`  
`BEGIN`  
`table[i,j,k] := i + j + k - 2;`  
`writeln(i:4, j:2, k:3, table[i,j,k]:4)`  
`END;`  
`.`  
`.`  
`END.`
- (p) `PROGRAM sample(input,output);`  
`VAR buffer : ARRAY [1..80] OF char;`  
`target : PACKED ARRAY [1..80] OF char;`  
`BEGIN`  
`.`  
`.`  
`pack(buffer,target,5);`  
`.`  
`.`  
`unpack(target,12,buffer);`  
`.`  
`.`  
`END.`

```

(q) PROGRAM sample(input,output);
VAR line : PACKED ARRAY [1..80] OF char;
    name : PACKED ARRAY [1..40] OF char;
    address : PACKED ARRAY [1..40] OF char;
BEGIN
    .
    .
    IF name <> 'end' THEN line := name + address;
    .
    .
END.

(r) PROGRAM sample(input,output);
VAR foreground,background : PACKED ARRAY [1..5] OF char;
    flag : boolean;
BEGIN
    .
    .
    background := 'black';
    .
    .
    IF flag THEN foreground := background
    ELSE foreground := 'green';
    .
    .
END.

```

(٤٥) فيما يلي عدة تخطيطات هيكلية توضح استخدام منظومات أو عناصر منظومات مع إجراءات أو نوال ، بعضها مكتوب بطريقة خاطئة ، عرف كل الأخطاء .

```

(a) PROGRAM sample(input,output);
VAR list : ARRAY [1..100] OF integer;

PROCEDURE process (dummy : ARRAY [1..100] OF integer);
BEGIN
    .
    .
    .
END;

BEGIN (* main action statements *)
    .
    .
    process(list);
    .
    .
END.

```

```
(b) PROGRAM sample(input,output);
    TYPE list = ARRAY [1..100] OF real;
    VAR list1,list2 : list;

    PROCEDURE module (VAR x : list);
    BEGIN
        .
        .
        (* process the elements of x *)
        .
    END;

    BEGIN (* main action statements *)
        .
        .
        module(list1);
        .
        .
        module(list2);
        .
        .
    END.

(c) PROGRAM sample(input,output);
    VAR sales : ARRAY [1..100] OF integer;
    result : integer;

    FUNCTION funct1 (x : integer) : integer;
    BEGIN
        .
        .
        .
        funct1 := . . .
    END;

    BEGIN (* main action statements *)
        .
        .
        result := funct1 (sales [6] + sales [12]);
        .
        .
    END.

(d) PROGRAM sample(input,output);
    VAR line : PACKED ARRAY [1..80] OF char;

    PROCEDURE proc1 (dummy : char);
    BEGIN
        .
        .
        .
    END;

    BEGIN (* main action statements *)
        .
        .
        proc1(line[40]);
        .
        .
    END.
```

- 
- (e) `PROGRAM sample(input,output);`  
`VAR list : ARRAY [1..100] OF integer;`  
`items : ARRAY [1..25] OF integer;`
- `PROCEDURE process(x : ARRAY [first..last : integer] OF integer);`  
`BEGIN`  
`.`  
`.`  
`.`  
`END;`
- `BEGIN (* main action statements *)`  
`.`  
`.`  
`process(list);`  
`.`  
`.`  
`process(items);`  
`.`  
`.`  
`END.`
- (f) `PROGRAM sample(input,output);`  
`VAR stockno : ARRAY [1..100] OF integer;`  
`cost : ARRAY [1..100] OF real;`
- `PROCEDURE process(x : ARRAY [first..last : integer] OF integer);`  
`BEGIN`  
`.`  
`.`  
`.`  
`END;`
- `BEGIN (* main action statements *)`  
`.`  
`.`  
`process(stockno);`  
`.`  
`.`  
`process(cost);`  
`.`  
`.`  
`END.`
- (g) `PROGRAM sample(input,output);`  
`VAR page : PACKED ARRAY [1..66, 1..80] OF char;`
- `FUNCTION count (text : PACKED ARRAY`  
`[first..last : integer; start..stop : integer] OF char) : integer;`  
`BEGIN`  
`.`  
`.`  
`.`  
`count := . . .`  
`END;`

```

BEGIN  (* main action statements *)
.
.
  (* read the elements of text *)
.
.
  writeln(count(text));
.
.
END.

```

(h) PROGRAM sample(input,output);  
 VAR name : PACKED ARRAY [1..80] OF char;

```

    PROCEDURE proc1 (line : PACKED ARRAY [first..last : integer] OF char);
    BEGIN
      .
      .
      writeln(line);
      .
      .
    END;

  BEGIN  (* main action statements *)
  .
  .
  (* read the elements of name *)
  .
  .
  proc1(name);
  .
  .
  END.

```

(i) PROGRAM sample(input,output);  
 VAR item : ARRAY [1..100] OF real;

```

    PROCEDURE process (numbers : ARRAY [start..stop : integer] OF real);
    BEGIN
      .
      .
      .
    END;

    PROCEDURE module (x : ARRAY [start..stop : integer] OF real);
    BEGIN
      .
      .
      process(x);
      .
      .
    END;

  BEGIN  (* main action statements *)
  .
  .

```



```

        module(item);
        .
        .
        process(item);
        .
        .
    END.

(j) PROGRAM sample(input,output);
    VAR a : ARRAY [1..20] OF real;
        b : ARRAY [1..50] OF integer;

    PROCEDURE demo (VAR x,y : ARRAY [first..last : integer] OF real);
    BEGIN
        .
        .
        .
    END;

    BEGIN (* main action statements *)
        .
        .
        demo(a,b);
        .
        .
    END.

```

## Programming Problems

مشاكل برمجة :

(٤٦) عدل البرنامج المعطى فى مثال ٩ - ٨ ، بحيث يعاد ترتيب الأعداد فى تسلسل جبرى منخفض القيم ( أى من الأكبر إلى الأصغر ) . اختبر البرنامج مستخدما البيانات المعطاه فى مثال ٩ - ٨

(٤٧) عدل البرنامج المعطى فى مثال ٩ - ٨ بحيث يمكن تنفيذ إعادة ترتيب أى مما يلى :

(أ) من الأصغر إلى الأكبر بالنسبة للقيمة .

(ب) من الأصغر إلى الأكبر جبريا .

(ج) من الأكبر إلى الأصغر بالنسبة للقيمة .

(د) من الأكبر إلى الأصغر جبريا .

مع تواجد قائمة تسمح للمستخدم أن يختار أى إعادة ترتيب يريد استخدامها فى كل مرة يقوم بتنفيذ البرنامج .  
اختبر البرنامج مستخدما القيم العشر التالية .

4.7	-8.0
-2.3	11.4
12.9	5.1
8.8	-0.2
6.0	-14.7

(٤٨) عدل البرنامج المعطى فى مثال ٩ - ١٦ لحساب الفروق ، بدلا من المجموع للعناصر المتناظرة فى كل من الجدولين . اختبر البرنامج مستخدما البيانات المعطاه فى مثال ٩ - ١٦ .

(٤٩) عدل البرنامج المعطى فى مثال ٩ - ٧ بحيث إنه يستخدم منظومات مضغوطة ، ومؤشرات منظومة متغيرة الطول . اختبر البرنامج مستخدما اسمك وعنوانك .

(٥٠) عدل برنامج pig latin الموجود فى مثال ٩ - ٢٧ ، بحيث يمكن احتواء علامات التنقيط ، والحروف الكبيرة ، والأصوات مزدوجة الحروف .

(٥١) اكتب برنامجا بلغة البسكال ، يدخل سطرا من أحد النصوص ، ويخزنه فى منظومة مضغوطة ، ثم يكتبها بعد ذلك . اجعل طول السطر غير محدد ( ويفصل بعودة العربة ) ، مقترضا أنها لن تزيد عن 80 خانة . اختبر البرنامج مستخدما أى سطر تعدده بنفسك . قارن البرنامج ببرنامج مثال ٧ - ٢٤ الذى يستخدم إجراء إعادة ، بدلا من استخدام منظومة . حدد أى طريقة هى الأفضل .

(٥٢) فيما يلى مجموعة مشاكل برمجة تستخدم درجات الطلبة التالية : ( الدرجات لستة امتحانات أداها كل طالب فى مقرر لغة البسكال ) .

Name	Exam scores, percent					
Adams	45	80	80	95	55	75
Brown	60	50	70	75	55	80
Davis	40	30	10	45	60	55
Fisher	0	5	5	0	10	5
Hamilton	90	85	100	95	90	90
Jones	95	90	80	95	85	80
Ludwig	35	50	55	65	45	70
Osborne	75	60	75	60	70	80
Prince	85	75	60	85	90	100
Richards	50	60	50	35	65	70
Smith	70	60	75	70	55	75
Thomas	10	25	35	20	30	10
Wolfe	25	40	65	75	85	95
Zorba	65	80	70	100	60	95

(١) اكتب برنامجا تقليديا بلغة البسكال يقبل اسم كل طالب ودرجاته كمدخلات ، وحدد متوسط الدرجات لكل طالب ، ثم أخرج اسمه مع درجات كل الامتحانات والمتوسط المحسوب . ( ملاحظة : لاحظ أن كل اسم يتبعه فراغ ) . اجعل البرنامج عاما بقدر الإمكان ( انظر المشكلة رقم ٥٠ فى الفصل السادس الجزء k ) .

(ب) عدل البرنامج المكتوب للمشكلة السابقة ليسمح بترجيح غير متساو لدرجات الامتحانات . افرض بصفة خاصة أن كل امتحان من الامتحانات الأربعة الأولى يسهم بمقدار ١٥ ٪ من الدرجة النهائية ، وأن كل امتحان من الامتحانين الآخرين يمثل ٢٠ ٪ من الدرجة النهائية ( انظر المشكلة رقم ٥٠ فى الفصل السادس الجزء L ) .

(ج) وسع البرنامج المكتوب للمشكلة السابقة ، بحيث تحسب متوسط درجة الفصل ، بالإضافة إلى متوسطات الطلاب ( انظر المشكلة رقم ٥٠ فى الفصل السادس الجزء m ) .

(د) وسع البرنامج المكتوب للمشكلة السابقة ، بحيث يحسب انحراف متوسط درجة كل طالب من متوسط درجة الفصل . اكتب كمخرجات متوسط درجة الفصل ، يتبعها اسم كل طالب ، ودرجاته ، ومتوسطها ، والانحراف عن متوسط درجة الفصل . تأكد أن المخرجات مرتبة منطقيا ، ولها عناوين واضحة .

(٥٣) اكتب برنامجا بلغة البسكال ، ينتج جدولا يقيم المعادلة التالية :

$$y = 2e^{-0.1t} \sin 0.5t$$

حيث  $t$  تتغير من 0 إلى 60 . أسمح بزيادة تحدث في  $t$  ، ويتم إدخالها كمؤشر مدخلات .

(٥٤) اكتب برنامجا بلغة البسكال ينتج جدولا لمعامل الفائدة المركبة  $(F/P)$  ، حيث :

$$A/P = \frac{i(1+i)^n}{(1+i)^n - 1}$$

تمثل  $F$  في هذه العلاقة القيمة المستقبلية لمبلغ معين من المال . وتمثل  $P$  قيمته الحالية . كما تمثل  $i$  معدل الفائدة السنوية كنسبة مئوية . وتمثل  $n$  عدد السنوات . دع كل صف في الجدول يناظر قيمة مختلفة من قيم  $n$  ، تتراوح قيم  $n$  من 1 إلى 30 ( أى أنه هناك 30 سطرا ) . ودع كل عمود يمثل معدل فائدة مختلف . الجدول يحتوى على معدلات فائدة : 4 , 4.5 , 5 , 5.5 , 6 , 6.5 , 7 , 7.5 , 8 , 8.5 , 9 , 9.5 , 10 , 11 , 12 , 15 فى المائة ( أى هناك 16 عمودا ) تأكد من وضع أسماء مناسبة للصفوف والأعمدة .

(٥٥) اكتب برنامجا بلغة البسكال يعيد ترتيب قائمة بها كلمات ترتيبيا أبجديا . لعمل ذلك أدخل الكلمات فى منظومة مضغوطة ذات بعدين من النوع الحرفى ، على أن يمثل كل صف كلمة كاملة . بعد ذلك يمكن إعادة ترتيب الصفوف بنفس الطريقة التى أعيد ترتيب قائمة بالأعداد بها من العدد الأصغر إلى العدد الأكبر ( انظر مثال ٩ - ٨ ) .

استخدم البرنامج ليعيد ترتيب الأسماء التالية ، وكن حذرا عند استخدام الحروف الأولى من الأسماء .

Washington	Polk	Arthur	Roosevelt, F. D.
Adams, J.	Taylor	Cleveland	Truman
Jefferson	Fillmore	Harrison, B.	Eisenhower
Madison	Pierce	McKinley	Kennedy
Monroe	Buchanan	Roosevelt, T.	Johnson, L. B.
Adams, J. Q.	Lincoln	Taft	Nixon
Jackson	Johnson, A.	Wilson	Ford
Van Buren	Grant	Harding	Carter
Harrison, W. H.	Hayes	Coolidge	Reagan
Tyler	Garfield	Hoover	

(٥٦) اعتبر القائمة التالية التى تحتوى على أسماء بلاد وأسماء عواصم .

Canada	Ottawa	Israel	Jerusalem
England	London	Italy	Rome
France	Paris	Japan	Tokyo
India	New Delhi	Mexico	Mexico City

People's Republic of China	Peking
United States	Washington
U.S.S.R.	Moscow
West Germany	Bonn

اكتب برنامجا بلغة البسكال متداخلا ، يقبل اسم البلد كمدخلات ، ويكتب العاصمة المناظرة لها ، والعكس كذلك .  
صمم البرنامج ، بحيث يستمر حتى تظهر كلمة end كمدخلات .

(٥٧) اكتب برنامجا كاملا بلغة البسكال لكل مشكلة من المشاكل المذكورة أدناه . حدد نوع المنظومة الأكثر ملاءمة لكل مشكلة . تأكد من استخدام الأجزاء في كل برنامج ، ومن كتابة عناوين واضحة للمخرجات ، ومن استخدام أنواع طبيعية للبيانات ومكونات تحكم ذات كفاءة .

(١) افرض أنه معطى لنا جدول من الأعداد الصحيحة A ، به m صف ، و n عمودا ، ولدينا قائمة بالأعداد الصحيحة X ، بها n عنصرا . ونريد إنتاج قائمة أعداد صحيحة جديدة Y ، يتم إعدادها عن طريق أداء العمليات التالية :

$$\begin{aligned} Y[1] &= A[1,1]*X[1] + A[1,2]*X[2] + \dots + A[1,N]*X[N] \\ Y[2] &= A[2,1]*X[1] + A[2,2]*X[2] + \dots + A[2,N]*X[N] \\ &\vdots \\ Y[M] &= A[M,1]*X[1] + A[M,2]*X[2] + \dots + A[M,N]*X[N] \end{aligned}$$

اكتب كمخرجات بيانات المدخلات ( أى قيم العناصر A , X ) . يليها قيم عناصر Y .

استخدم البرنامج في تشغيل البيانات التالية :

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \end{bmatrix}$$

$$X = \begin{bmatrix} 1 \\ -8 \\ 3 \\ -6 \\ 5 \\ -4 \\ 7 \\ -2 \end{bmatrix}$$

(ب) افرض أن  $A$  جدول من الأعداد الحقيقية ، به  $K$  صفاً ، وبه  $m$  عموداً ، وأن  $B$  جدول من الأعداد الحقيقية ، به  $m$  صفاً ، وبه  $n$  عموداً . ونريد إنتاج جدولاً جديداً  $C$  ، حيث يتحدد كل عنصر من عناصر الجدول  $C$  بالعلاقة التالية :

$$C[i,j] = A[i,1]*B[1,j] + A[i,2]*B[2,j] + \dots + A[i,m]*B[m,j]$$

حيث :

$$i = 1, 2, \dots, k$$

$$\text{and } j = 1, 2, \dots, n.$$

اكتب عناصر الجداول الثلاثة  $A$  ,  $B$  ,  $C$  ، وتأكد من وجود عناوين واضحة لكل شيء .

استخدم البرنامج فى تشغيل مجموعة البيانات التالية :

$$A = \begin{bmatrix} 2 & -1/3 & 0 & 2/3 & 4 \\ 1/2 & 3/2 & 4 & -2 & 1 \\ 0 & 3 & -9/7 & 6/7 & 4/3 \end{bmatrix}$$

$$B = \begin{bmatrix} 6/5 & 0 & -2 & 1/3 \\ 5 & 7/2 & 3/4 & -3/2 \\ 0 & -1 & 1 & 0 \\ 9/2 & 3/7 & -3 & 3 \\ 4 & -1/2 & 0 & 3/4 \end{bmatrix}$$

(ج) اعتبر تسلسلاً من الأعداد الصحيحة  $x_i$  ، حيث  $i = 1, 2, \dots, m$  . ويعرف المتوسط بانه :

$$\bar{x} = \frac{(x_1 + x_2 + \dots + x_m)}{m}$$

والانحراف عن المتوسط هو :

$$d_i = (x_i - \bar{x}), \quad i = 1, 2, \dots, m$$

والانحراف المعياري هو :

$$s = \sqrt{\frac{(d_1^2 + d_2^2 + \dots + d_m^2)}{m}}$$

أدخل أول  $m$  عنصراً من منظومة حقيقية ذات بعد واحد ، احسب مجموع هذه العناصر ومتوسطها وانحرافها عن المتوسط والانحراف المعياري لها ، وأكبر قيمة جبرية ، وأصغر قيمة جبرية . استخدم البرنامج فى تشغيل مجموعة البيانات التالية :

27.5	87.0
13.4	39.9
53.8	47.7
29.2	8.1
74.5	63.2

كرر الحسابات لعدد  $k$  من قوائم الأعداد المختلفة . احسب المتوسط الكلى ، والانحراف المعياري الكلى ، وأقصى قيمة مطلقة ، وأقل قيمة مطلقة .

(د) افرض أنه لدينا مجموعة من القيم المجدولة لكل من  $x$  ,  $y$  كما يلي :

$$\begin{array}{ccccccc} y_0 & y_1 & y_2 & \dots & y_n \\ x_0 & x_1 & x_2 & \dots & x_n \end{array}$$

وأننا نريد الحصول على قيمة  $y$  التى تتاظر  $x$  معينة ، وتقع بين قيمتين من قيم  $y$  الموجودة فى الجدول . وعادة ما تحل هذه المشكلة باستخدام الاستدلال interpolation ، أى بعمل كثيرة حدود  $y(x)$  تمر خلال  $n$  نقطة ، مثل :

$$y(x_0) = y_0, y(x_1) = y_1, \dots, y(x_n) = y_n$$

ثم تقويم  $y$  عند قيمة  $x$  المعطاه . وتستخدم صيغة لاجرانج Lagrange form لأداء هذا الاستدلال على كثيرة الحدود . ولعمل ذلك ... فإننا نكتب :

$$y(x) = f_0(x)*y_0 + f_1(x)*y_1 + \dots + f_n(x)*y_n$$

حيث  $f_i(x)$  هى كثيرة حدود ، بحيث إن :

$$f_i(x) = \frac{(x-x_0)(x-x_1)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_n)}{(x_i-x_0)(x_i-x_1)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_n)}$$

لاحظ أن :  $f_i(x_i) = 1$  ,  $f_i(x_j) = 0$  ,

حيث  $x_i$  هى قيمة مجدولة من قيم  $x$  تختلف عن  $x_i$  . وعلى هذا ... فإننا نتأكد من أن  $y(x_i) = y_i$  .

اكتب برنامجاً بلغة البسكال لقراءة  $n$  زوجاً من البيانات ، حيث  $n$  لا تزيد عن 10 ، ثم الحصول على قيمة الاستدلال لـ  $y$  عند قيمة أو أكثر من قيم  $x$  . استخدم البرنامج فى الحصول على قيم  $y$  المحسوبة من الاستدلال عند قيم  $x=112$  ,  $x=174$  ,  $x=13.7$  من البيانات المذكورة أدناه . حدد عدد أزواج البيانات المجدولة المطلوبة فى كل حساب ، بغرض الحصول على قيمة لـ  $y$  ذات دقة معقولة .

(٥٨) اكتب برنامج بسكال متداخل يعد شفرة لسطر نصى أوفيك الشفرة .

لعمل شفرة للسطر النصى نفذ مايلي :

- ١ - حول كل رمز ، بما فى ذلك الفراغات ، الى مكافئة من شفرة ASCII .
- ٢ - انتج رقماً عشوائياً موجباً . أضف هذا الرقم الى مكافئ شفرة ASCII لكل رمز . ( يستخدم نفس الرقم العشوائى لبقية محتويات السطر النصى ) .
- ٣ - افرض أن N1 يمثل أقل قيمة مسموح بها فى شفرة ASCII ، ويمثل N2 أعلى قيمة مسموح بها . فإذا كان الرقم الذى يتم الحصول عليه فى الخطوة رقم 2 ( أى مكافئ ASCII الاصلى مضافاً اليه الرقم العشوائى ) أكبر من N2 ويمثل دائماً ، على هذا ، أحد رموز ASCII .
- ٤ - أطبع الرموز المناظرة لقيم ASCII المشفرة .

عندما يراد فك الشفرة لسطر النصى تعكس الخطوات . كن متأكداً على أية حال من استخدام نفس الرقم العشوائى فى فك الشفرة والذى سبق استخدامه فى عمل الشفرة .

## الفصل العاشر

### السجلات

# Records

لقد درسنا في الفصل السابق المنظومة ، وهي نوع من البيانات المرتبة التي يجب أن تكون كل عناصرها من نفس النوع . ونعيد انتباهنا الآن إلى نوع آخر مهم من أنواع البيانات المرتبة ، وهو السجل record الذي لا تحتاج العناصر المكونة له أن تكون من نفس النوع . وعلى هذا فيمكننا أن نشير إلى مجموعة من عناصر بيانات تختلف في النوع عن بعضها . ويقال لعناصر البيانات الفردية أنها حقول fields داخل السجل .

#### ١ - تعريف السجل : 1. DEFINING A RECORD

أسهل طريقة لتعريف السجل هي التعريف كجزء من توضيح المتغيرات ، وهذا يسمح لنا بتوضيح متغير فردي للسجل ، كما تعرف المتغيرات البسيطة ومتغيرات المنظومات .

والصيغة العامة لتوضيح متغير السجل هي .

```
VAR record name : RECORD field 1; field 2; . . .;field n END
```

حيث تمثل field 1 توضيح أول حقل ، وتمثل field 2 الحقل الثاني ، وهكذا . ويكتب توضيح كل حقل بطريقة تشبه توضيح المتغير الفردي ، أي على الصورة التالية :

*field name : type*

حيث field name هو معرف يمثل اسم الحقل ، و type هو نوع بيانات عنصر البيانات التي تشغل الحقل .

لاحظ أن الحقول الفردية داخل السجل مفصولة عن بعضها بواسطة فواصل منقوطة .

مثال (١٠-١)

افترض أن سجل العميل يحتوى على رقم العميل من النوع الصحيح ، وتحديد لحساب العميل من النوع الحرفي وموازنة العميل ، وهي عدد حقيقي . دعنا نمثل هذا السجل بمتغير سجل يسمى customer ، وتوضيح المتغير هو مايلي :

```
VAR customer : RECORD
    custno : integer;
    custtype : char;
    custbalance : real
END;
```

لاحظ أن السجل يحتوى على حقل صحيح النوع اسمه custno ، وحقل من النوع الحرفي اسمه cust type ، وحقل من النوع الحقيقي اسمه Custbalance .

ولاحتاج توضيحات الحقول الفردية أن تكتب على أسطر منفردة دائما كما في هذا المثال . وعادة ماتكتب على أية حال بهذه الطريقة لتسهيل القراءة .

وهناك طريقة أخرى لتعريف السجل ، وعادة ماتكون أكثر نفعا من توضيح المتغير ، وهي تعريف نوع type السجل . وبعد ذلك يمكن توضيح أن المتغيرات الفردية من هذا النوع . لاحظ التعامل مع الطرق المستخدمة في تعريف المنظومات ، والتي سبق ذكرها في الفصل السابق . وفي الصورة العامة يكتب تعريف نوع السجل على النحو التالي :

```
TYPE name = RECORD field 1; field 2; . . .;field n END
```

حيث إن معنى العناصر الفردية هو نفسه كما سبق ذكره .

### مثال (١٠-٢)

اعتبر مرة أخرى سجل العميل المذكور في المثال السابق . دعنا نعرف الآن نوع سجل اسمه account ، ومتغير مناظر اسمه customer كما سبق ذكره . يمكن تحقيق ذلك بكتابة

```
TYPE account = RECORD
    custno : integer;
    custtype : char;
    custbalance : real
END;
VAR customer : account;
```

يجب أن يكون مفهوما أن هذا يحقق نفس الشيء ، مثل توضيح المتغير الموضح في مثال ١٠ - ١ . الطريقة الحالية عامة أكثر ، لأنها تسمح لنا بإدخال متغيرات سجلات إضافية إذا ما أردنا ذلك .

ويمكن أن يصاحب الحقول الفردية أنواع بيانات يحددها المستفيد ، وأنواع مدى جزئي ، وأنواع بيانات قياسية . أكثر من هذا ... يمكن ترتيب أنواع الحقول المدرجة Scalar . وعلى هذا ... يمكن لعنصر فردي أن يكون منظومة أو أي سجل آخر .

### مثال (١٠-٣)

فيما يلي تعريف آخر للسجل الموضح في مثال ١٠ - ٢

```
TYPE status = (current,overdue,delinquent);
account = RECORD
    custno : 1..9999;
    custtype : status;
    custbalance : real
END;
VAR customer : account;
```

رقم العميل custno هنا عبارة عن عنصر بيانات من نوع المدى الجزئي ، ونوع العميل cust type هو عنصر بيانات من النوع المتعدد .



لاحظ أن عنصر البيانات من النوع المتعدد ( Status ) يجب أن يعرف بنفسه قبل أن يشار إليه في توضيح السجل .

### مثال (١٠-٤)

دعنا نضيف الآن اسم العميل إلى السجل الذي سبق ذكره . لعمل ذلك ... فإننا نعرف حقلاً إضافياً يمثل منظومة مضغوطة من النوع الحرفي . وعلى هذا تصبح تعريف محتويات السجل على النحو التالي :

```
TYPE status = (current,overdue,delinquent);
account = RECORD
    custname : PACKED ARRAY [1..80] OF char;
    custno : 1..9999;
    custtype : status;
    custbalance : real
END;
VAR customer : account;
```

وفيما يلي تعريفاً آخر لهذا السجل الذي يعتبر عاماً بعض الشيء .

```
TYPE status = (current,overdue,delinquent);
line = PACKED ARRAY [1..80] OF char;
account = RECORD
    custname : line;
    custno : 1..9999;
    custtype : status;
    custbalance : real
END;
VAR customer : account;
```

يسمح هذا التعريف بتعريف معلومات من نوع السلسلة الأخرى بسهولة جداً ، وذلك داخل السجل الحالي ، أو في أي جزء آخر من أجزاء البرنامج . وعلى هذا ... فإذا ما أردنا أن نضيف حقلاً عنوان إلى customer ، فيمكننا كتابة مايلي :

```
TYPE status = (current,overdue,delinquent);
line = PACKED ARRAY [1..80] OF char;
account = RECORD
    custname : line;
    custaddress : line;
    custno : 1..9999;
    custtype : status;
    custbalance : real
END;
VAR customer : account;
```

### مثال (١٠-٥)

دعنا نضيف الآن ( تاريخ ) آخر مبلغ دفعه العميل إلى سجل العميل . ويمكن أن يتحقق ذلك عن طريق تمثيل

التاريخ كسجل منفصل ، ثم اعتبار هذا السجل كحقل من حقول customer . وعلى هذا ... يصبح السجل الآن على النحو التالي :

```
TYPE status = (current,overdue,delinquent);
line = PACKED ARRAY [1..80] OF char;
date = RECORD
    month : 1..12;
    day : 1..31;
    year : 1900..2100
END;
account = RECORD
    custname : line;
    custaddress : line;
    custno : 1..9999;
    custtype : status;
    custbalance : real;
    lastpayment : date
END;
VAR customer : account;
```

لاحظ أن السجل account يحتوى الآن على سجل آخر ، وهو lastpayment كأحد حقوله . كما نرى أيضا أن account يحتوى على منظومتين ( Custname , Custaddress ) ونوع مدى جزئى ( Custno ) ونوع بيانات متعددة ( Custtype ) ونوع بيانات قياسي ( Custbalance ) . وعلى هذا فإن هذا المثال يوضح المرونة الموجودة فى تعريف السجل .

ويجب أن يكون معروفا أيضا أنه يمكن للسجل أن يكون عنصر بيانات فردياً فى نوع بيانات مرتب آخر مثل المنظومة . وعلى هذا يمكننا تعريف منظومات عناصرها سجلات وسجلات عناصرها ( أو بعض من عناصرها ) منظومات ، وما إلى ذلك .

مثال (١٠-٦)

فيما يلى مثالا لمنظومة وعناصرها سجلات . وبصفة خاصة دعنا نعرف منظومة اسمها event ، وعناصرها الفردية هى data كما سبق تعريفها فى المثال السابق .

```
TYPE date = RECORD
    month : 1..12;
    day : 1..31;
    year : 1900..2100
END;
VAR events : ARRAY [1..100] OF date;
```

وعلى هذا يكون لدينا منظومة ذات بعد واحد ( وهى list ) يمكن أن تحتوى على تواريخ dates مختلفة ، تصل إلى ١٠٠ تاريخ ، لاحظ أن events معرف بأنه متغير ، وليس نوع بيانات .

وبعد تعريف نوع سجل فردى ، يمكننا توضيح متغيرات عديدة مختلفة كسجلات من هذا النوع . وهذه السمة موضحة فى المثال التالى .

## مثال (١٠-٧)

اعتبر السجل account المعرف في المثال رقم (١٠-٥) . يمكننا الآن توضيح المتغيرين regular , preferred بأنهما سجلات من هذا النوع .

وعلى هذا نكتب مايلي :

```
TYPE status = (current,overdue,delinquent);
line = PACKED ARRAY [1..80] OF char;
date = RECORD
    month : 1..12;
    day : 1..31;
    year : 1900..2100
END;
account = RECORD
    custname : line;
    custaddress : line;
    custno : 1..9999;
    custtype : status;
    custbalance : real;
    lastpayment : date
END;
VAR preferred,regular : account;
```

ويمكن ضغط السجل كما هو الحال مع ضغط المنظومات تماما . ويسمح ذلك بتخزين السجل في صورة أكثر إيجازا ، بالرغم من أن الاتصال بعنصر بيانات داخل مثل هذا السجل يستغرق وقتا أطول .

ويتحقق ضغط السجلات بكتابة PACKED RECORD ، بدلا من RECORD في تعريف نوع السجل كما هو موضح في المثال التالي .

## مثال (١٠-٨)

دعنا نعيد تعريف نوع السجل account ، والذي ظهر في المثال (١٠-٢) لكي يكون سجلا مضغوطا .

```
TYPE account = PACKED RECORD
    custno : integer;
    custtype : char;
    custbalance : real
END;
VAR customer : account;
```

وعلى هذا فإن المتغير Customer يمثل سجلا مضغوطا من نفس نوع السجل account .

ويشمل ضغط السجلات ، مثل ضغط المنظومات ، موازنة بين سرعة التنفيذ ومتطلبات الذاكرة . وعلى أية حال فإن هذه السمة ليست متسعة الانتشار في استخدامها مثل سمة ضغط المنظومات . وكقاعدة عامة ... تستخدم هذه السمة في التطبيقات التي تركز على نسخ محتويات السجلات فقط ، وذلك بدلا من التطبيقات الأكثر شيوعا ، والتي يعامل فيها عناصر فردية من عناصر السجلات .

وأخيرا يجب أن يذكر أن كل سجل يعتبر كبنية مستقلة بذاتها بالنسبة لتعريفات الحقول . وعلى هذا ... يجب أن يكون لكل حقل داخل أى سجل اسم فريد ، إلا أن نفس هذا الاسم الفريد فى السجل يمكن استخدامه فى سجلات أخرى . وفى كلمات أخرى يضيق مدى معرف الحقل ليشمل السجل المحدد المعروف داخله فقط .

مثال (١٠-٩)

فيما يلي تعريفًا لنوعين مختلفين one و two .

```
TYPE one = RECORD
    a : real;
    b : integer;
    c : char
END;
= RECORD
    a : char;
    b,c : real
END;
```

لاحظ أن أسماء الحقول الفردية a , b , c مكررة فى السجلين ، إلا أن أنواع البيانات المصاحبة لها مختلفة ، وهذا مسموح به ، حيث إن مدى كل مجموعات تعريف الحقول يشمل السجل المناظر له فقط . لاحظ أيضا أن أسماء الحقول مميزة داخل كل سجل طبقا لما هو مطلوب .

## 2. PROCESSING A RECORD

### ٢ - تشغيل السجل :

والآن ... وبعد أن رأينا كيف تعرف السجلات ، دعنا نعيد انتباهنا لاستخدام السجلات ، أو استخدام مكونات السجلات فى مواقف برمجية تقليدية .

ويشمل أبسط أنواع عمليات تشغيل السجلات تحديد أحد محتويات السجل الآخر . ويتطلب ذلك أن يكون السجلان لهما نفس التكوين تماما .

مثال (١٠ - ١٠)

فيما يلي تخطيطا هيكليا لبرنامج بسكال ، يحتوى على تحديد سجل لآخر ( لاحظ أن هذا التخطيط يستخدم تعريفات السجلات التى سبق تقديمها فى المثال رقم (١٠ - ٨) ) .

```
TYPE status = (current,overdue,delinquent);
line = PACKED ARRAY [1..80] OF char;
date = RECORD
    month : 1..12;
    day : 1..31;
    year : 1900..2100
END;
```

```

account = RECORD
    custname : line;
    custaddress : line;
    custno : 1..9999;
    custtype : status;
    custbalance : real;
    lastpayment : date
END;
VAR preferred,regular : account;

BEGIN
    .
    .
    preferred := regular;
    .
    .
END.

```

ويقتضى بالطبع أن عناصر regular تم إدخالها داخل الكمبيوتر ، أو أنها سبق تعريفها قبل عبارة التحديد .

ومن الشائع أكثر تشغيل عناصر السجل الفردية ، بدلا من تشغيل محتويات السجل مع بعضها . ولعل ذلك يجب أن نكون قادرين على الاتصال بعناصر السجل الفردية . ويتحقق ذلك عن طريق عمل لقب ( أو محدد ) للحقل field designator ، وهو عبارة عن خليط من اسم متغير من نوع السجل ، واسم حقل . أى أن :

*variable name.field name*

( لاحظ وجود النقطة بين الاسمين ) .

مثال (١٠-١١)

اعتبر مرة أخرى المتغيرين من نوع السجل preferred , regular المعرفين في مثال (١٠ - ١٠) . يشار إلى عناصر البيانات الفردية الموجودة داخل كل من هذه السجلات بمحدداته للحقل مثل :

regular . custno , preferred. custname , regular. lastpayment

فمثلا preferred. custaddress, preferred. custname يمثلان متغيرات سلسلة ، كل منها به ٨٠ عنصرا ، ويمثل preferred. custno كمية عددية صحيحة تقع قيمتها بين 1,9999 ويمثل preferred.custtype إحدى الكميات البسيطة متعددة النوع من current , overdue , delinquent . كما يمثل preferred. custbalance كمية عددية حقيقية يعرفها تاريخ السجل ( المزيد من هذا موجود في المثال التالي ) . ويستخدم أوصاف حقول شبيهه لعناصر regular .

إذا مثل أحد الحقول عنصر بيانات مرتب ، فإن العناصر الفردية لهذا الحقل يمكن الاتصال بها بوضع عنصر البيانات المرتب في محدد الحقل . وعلى هذا فإذا مامل الحقل منظومة ، فيمكن الاتصال بأحد عناصر المنظومة بكتابة مايلي :

*variable name.field name [index values]*

وبالمثل إذا مامثل أحد الحقول سجلا محصورا ، فيمكن الاتصال بعنصر فردي من عناصر السجل المحصور كما يلي :

*variable name.field name.sub-field name*

حيث يشير sub - field إلى حقل داخل سجل محصور .

مثال (١٠-١٢)

اعتبر مرة أخرى المتغيرين من نوع السجل regular, preferred المعرفين في مثال (١٠ - ١٠) . يشير محدد الحقل [12] preferred.custname إلى الحرف الثاني عشر في أول حقل من preferred ( أى الحرف الثاني عشر في المنظومة المضغوطة custname ) .

وبالمثل يشير محدد الحقل regular.lastpayment.month إلى أول كمية عددية صحيحة في آخر حقل من regular ( أى الكمية الموجودة في أول حقل في السجل المحصور المسمى lastpayment ) . يجب أن تكون هذه الكمية عددية صحيحة ، تقع قيمتها داخل المدى الجزئي 1 .. حتى 12 كما يعرفها data ، وهو نوع السجل .

ويمكن استخدام عناصر السجل الفردية بنفس الطريقة مثل المتغيرات المعتادة . والمعالم الخاصة ( وكذلك القيود ) المطبقة على كل عنصر يتم تحديدها بنوع بيانات هذا العنصر . وعلى هذا ... إذا مامثل عنصر سجل كمية من النوع البسيط ، فيمكن أن يظهر على ذلك في عبارة تحديد أو في تعبير أو في مكون تحكم أو عبارة مدخلات أو مخرجات أو كمؤشر داخل إشارة إلى إجراء أو إلى دالة ، وما إلى ذلك طبقا للقيود المطبقة على نوع البيانات الخاص هذا .

وبالمثل إذا مامثل عنصر سجل نوع بيانات مرتب ( مثل المنظومة ، أو أى سجل آخر ) فيمكن استخدامه بنفس الطريقة ، مثل أى عنصر بيانات مرتب من نفس النوع . والقيود الخاص الوحيد هو أن أحد عناصر السجل المضغوط لا يمكن أن يمرر إلى إجراء أو إلى دالة إذا ما كان المؤشر الرسمي المناظر هو مؤشراً متغيراً .

مثال (١٠-١٣)

فيما يلي عدة عبارات غير مرتبطة ببعضها ، تستخدم عناصر سجل فردية . وكل عناصر السجل تؤيد تعريف السجل وتوضيحات متغيرات من نوع السجل المعطاء في مثال (١٠ - ١٠) ، وذلك مع افتراض التوافقية :

```
regular.custbalance := 0;

regular.custbalance := regular.custbalance - payment;

preferred.custname := regular.custname;

writeln(regular.custname, regular.custaddress);

average := (preferred.custbalance + regular.custbalance)/2;

newbalance := round(preferred.custbalance);

IF preferred.custname [1] = '*'
    THEN preferred.custtype := current;

IF regular.lastpayment.month < 6
    THEN writeln(regular.custno, regular.custbalance)
    ELSE preferred := regular;
```

تتطلب بعض التطبيقات أن يخزن تسلسل السجلات ويشغل بترتيب خاص . وفي مثل هذه الحالات يكون من المقنع تعريف منظومة ذات بعد واحد وعناصرها كلها سجلات . يمكن الاتصال على ذلك بسجل معين كما يلي :

*array name [index value]*

والاتصال بعنصر سجل معين كما يلي :

*array name [index value].field name*

مثال (١٠-١٤)

فيما يلي مثلاً للمنظومة ( list ) تحتوى على ١٠٠ سجلاً :

```
TYPE account = RECORD
    custno : integer;
    custtype : char;
    custbalance : real;
END;
VAR customer : ARRAY [1..100] OF account;
```

وعلى هذا يشير [23] customer إلى السجل رقم (٢٣) ، كما أن [23]. custbalance customer يشير إلى ثالث عنصر بيانات في السجل رقم (٢٣) ( وهو الموازنة الحالية للعميل رقم ٢٣ ) .

مثال (١٠-١٥)

فيما يلي مثلاً معقداً أكثر لعدة منظومات ( كل منظومة منها ) ذات بعد واحد ، وعناصرها سجلات .

```
TYPE status = (current,overdue,delinquent);
line = PACKED ARRAY [1..80] OF char;
date = RECORD
    month : 1..12;
    day : 1..31;
    year : 1900..2100;
END;
account = RECORD
    custname : line;
    custaddress : line;
    custno : 1..9999;
    custtype : status;
    custbalance : real;
    lastpayment : date;
END;
VAR preferred,regular : ARRAY [1..100] OF account;
```

في هذا المثال يشير [5] preferred إلى خامس سجل في المنظومة الخامسة ، ويشير [82] regular custname إلى اسم العميل في السجل رقم ٨٢ من المنظومة الثانية . وبالمثل يشير [14]. lastpayment regular إلى سنة آخر مبلغ دفعه العميل ( أى إلى العنصر الثالث في السجل المحصور ) في السجل رقم (١٤) من المنظومة الثانية .

### مثال (١٠-١٦)

فيما يلي عدة عبارات غير مرتبطة ببعضها ، تستخدم عناصر من سجلات من المثال السابق .

```
regular [82].custbalance := 0;
preferred [i].custbalance :=
    preferred [i].custbalance - payment;
writeln(regular [15].custname, regular [15].custaddress);
IF preferred [count].custname [1] = '*'
    THEN preferred [count].custtype := current;
IF regular [i].lastpayment.month < 6
    THEN writeln(regular [i].custno, regular [i].custbalance)
    ELSE preferred [i] := regular [i];
```

في هذه الأمثلة المتغيرات في i , count مفترض أنها متغيرات من النوع الصحيح ، محدد لها قيم مناسبة في جزء سابق في البرنامج .

### 3. THE WITH STRUCTURE

٣ - مكون WITH :

تتطلب برامج عديدة معاملة عناصر مختلفة من نفس السجل في أماكن مختلفة داخل البرنامج . ويمكن أن تصبح الحاجة إلى تحديد محددات مختلفة لحقول مملئة ، وعلى أية حال يمكن أن تقلل من سهولة قراءة البرنامج ككل . وفي مثل هذه الحالات يسمح مكون WITH بحذف أسم السجل من محددات الحقول .

والصيغة العامة لمكون WITH هي :

WITH record name DO statement

أو ، إذا ما وجد سجلان أو أكثر داخل المكون ، فإنها تصبح :

WITH record 1 name, record 2 name, . . . , record n name DO statement

جزء statement من المكون يشير إلى أى عبارة إجرائية ، والتي يمكن أن تكون هي بنفسها مكونا آخر . داخل هذه العبارة أى إشارة إلى عنصر داخل أحد السجلات المحددة لا تحتاج أن تحتوى على اسم السجل .

### مثال (١٠-١٧)

اعتبر التوضيح التالي :

```
TYPE date = RECORD
    month : 1..12;
    day : 1..31;
    year : 1900..2100
END;
VAR birthday : date;
```



أحدى الطرق لتعديل يوم الميلاد تُكتب :

```
BEGIN
    birthday.month := 5;
    birthday.day := 13;
    birthday.year := 1966
END;
```

من الأسهل على أية حال كتابة مايلى :

```
WITH birthday DO BEGIN month := 5; day := 13; year := 1966 END;
```

وكل من العبارتين يحقق نفس الشئ .

مثال (١٠-١٨)

فيما يلى مثالا آخر يوضح استخدام مكون WITH . يستخدم هذا المثال توضيحات سبق تقديمها فى مثال (١٠-٧) . وهذه التوضيحات معادة فى التخطيط الهيكلى التالى للبرنامج .

```
TYPE status = (current,overdue,delinquent);
line = PACKED ARRAY [1..80] OF char;
date = RECORD
    month : 1..12;
    day : 1..31;
    year : 1900..2100
END;
account = RECORD
    custname : line;
    custaddress : line;
    custno : 1..9999;
    custtype : status;
    custbalance : real;
    lastpayment : date
END;
VAR preferred,regular : account;

BEGIN
.
.
WITH regular DO
    BEGIN
        .
        .
        custno := 1262;
        lastpayment.day := 8;
        .
        .
        IF custbalance = 0 THEN custtype := current
            ELSE custtype := overdue;
        .
        .
    END;
.
.
END.
```

لاحظ أن كل إشارات الحقول داخل مكون WITH تشير إلى عناصر من regular . إذا كان ضروريا الإشارة إلى عنصر من عناصر preferred ، فإن محتوى محدد الحقل ( أى preferred , custbalance ) . يجب أن يحدد .

### مثال (١٠-١٩)

اعتبر التخطيط الهيكلي التالى للبرنامج الذى يستخدم توضيح المنظومة من نوع السجل الموضحة فى مثال (١٥-١٠) .

```

TYPE status = (current,overdue,delinquent);
line = PACKED ARRAY [1..80] OF char;
date = RECORD
    month : 1..12;
    day : 1..31;
    year : 1900..2100
END;

account = RECORD
    custname : line;
    custaddress : line;
    custno : 1..9999;
    custtype : status;
    custbalance : real;
    lastpayment : date
END;

VAR i : 1..100;
    preferred,regular : ARRAY [1..100] OF account;

BEGIN
.
.
FOR i := 1 TO 100 DO
    WITH regular [i] DO
        BEGIN
            .
            .
            custno := i + 1200;
            lastpayment.day := 8;
            .
            .
            IF custbalance = 0 THEN custtype := current
                ELSE custtype := overdue;
            .
            .
        END;
    .
.
END.

```

قارن هذا المثال مع التخطيط الموضح فى مثال (١٠-١٨) .

إذا ما احتوى مكون WITH على اسمى متغيرين أو أكثر من نوع السجل لها نفس أسماء الحقول ، فمن الممكن

ظهور إبهام في أى السجلات التى يشار إليها عند الاتصال بحقل داخل المكون . ويحل هذا الإبهام كما يلي :

مكون WITH متعدد السجلات :

WITH record 1 name, record 2 name, . . . , record n name DO statement

يُنظر المكون المتداخل التالى :

WITH record 1 name DO  
WITH record 2 name DO

WITH record n name DO statement

إذا ما كان اسم الحقل معتادا لسجلين أو أكثر من السجلات المحددة ، فإن مداه يتطابق مع السجل الداخلى جدا من التداخلات ، وأى إشارة إلى حقل داخل نفس الاسم ، لكنه يصاحب سجلا آخرًا يجب أن تحتوى على اسم السجل كجزء من محدد الحقل .

مثال (٢٠-١٠)

فيما يلي تخطيطا هيكليا لبرنامج :

```
VAR first : RECORD a,b,c : integer END;
    second : RECORD c,d,e : integer END;
BEGIN
.
.
WITH first, second DO
  BEGIN
    a := 1;
    b := 2;
    first.c := 3;
    c := 4;
    d := 5;
    e := 6;
  END;
.
.
END.
```

لاحظ أن c هى اسم حقل معتاد مع كل من السجلين . وعلى هذا ... فالإشارة إلى c داخل مكون WITH تشير تلقائيا إلى c الموجودة فى second ، حيث إن second هو اسم السجل الداخلى جدا . وبالإشارة إلى c الموجودة فى first داخل هذا المكون ، يجب أن نحدد محتوى محدد الحقل .

يتسبب المكون السابق فى تحديد القيم التالية للحقول المسماة c .

first.c := 3                      second.c := 4

ويجب استخدام المكون WITH فى الحالات التى يستخدم فيها بحرية . ويسهم استخدامه فى توضيح البرنامج ككل ، وفى بعض الحالات يمكن أن يحسن من كفاءة حسابات البرنامج .

## مثال (١٠-٢١)

نظام فواتير العملاء . اعتبر نظام فواتير بسيطة للعملاء ، حيث يتم إدخال سجلات العملاء فيه داخل الكمبيوتر ، ويتم تجديد موازنة كل عميل لتعكس مدفوعاته الحالية ، وحساب الموازنة الجديدة . ونحسب بالإضافة إلى ذلك الحالة الحالية لكل عميل ونعرضها طبقاً لموازنة العميل السابقة ولقيمة الموازنة الحالية .

افرض أن سجل كل عميل يحتوى على إحدى المعلومات التالية : الاسم - وعنوان الشارع - واسم المدينة - واسم الولاية - ورقم الحساب - وحالة الحساب ( حساب جارٍ أو متأخر أو مقصر ) . والموازنة السابقة والمدفوعات الحالية والموازنة الجديدة وتاريخ الدفع .

وفيما يلي توضيح السجل :

```
TYPE status = (current,overdue,delinquent);
line = PACKED ARRAY [1..30] OF char;
date = RECORD
    month : 1..12;
    day : 1..31;
    year : 1900..2100
END;
account = RECORD
    name : line;
    street : line;
    city : line;
    custno : 1..9999;
    custtype : status;
    oldbalance : real;
    newbalance : real;
    payment : real;
    paydate : date
END;
```

تحدد حالة كل حساب بالطريقة التالية : يعتبر الحساب جارياً ، إلا إذا كان :

١ - المبلغ المدفوع حالياً أكبر من صفر ، لكنه أقل ١٠ ٪ من الموازنة القائمة السابقة . يعتبر الحساب فى هذه إلى متأخر overdue .

٢ - يوجد موازنة قائمة ، والمبلغ المدفوع حالياً صفر ، وفى هذه الحالة يعتبر الحساب مقصراً delinquent .

وسوف تخزن كل السجلات كعناصر لمنظومة ذات بعد واحد .

وعلى هذا ... تصبح خطوات البرنامج الشامل كما يلي :

١ - حدد عدد الحسابات ( أى عدد السجلات ) التى سيتم تشغيلها .

٢ - اقرأ العناصر التالية لكل سجل :

أ - الاسم .

ب - الشارع .

ج - المدينة .

د - رقم الحساب .

هـ - الموزنة السابقة .

و - المبلغ المدفوع حاليا .

ز - تاريخ الدفع .

٣ - بعد قراءة كل السجلات داخل الكمبيوتر ، شغل كل سجل بالطريقة التالية :

١ - قارن المبلغ المدفوع حاليا مع الموزنة السابقة ، وحدد حالة الحساب المناسبة .

ب - احسب موازنة الحساب الجديدة بطرح المبلغ المدفوع حاليا من الموزنة السابقة ( وتحدد الموزنة السابقة الرصيد الدائن ) .

٤ - بعد تشغيل كل السجلات ، اكتب المعلومات التالية لكل سجل :

أ - الاسم .

ب - رقم الحساب .

ج - الشارع .

د - المدينة .

هـ - الموزنة القديمة .

و - القيمة المدفوعة حاليا .

ز - الموزنة الجديدة .

ح - حالة الحساب .

دعنا نكتب البرنامج على هيئة أجزاء مع ثلاثة إجراءات منفصلة لقراءة بيانات المدخلات ، وتشغيل البيانات ، وكتابة المخرجات على التوالي . وكل إجراء من هذه الإجراءات مباشر ، ولا يتطلب تفصيلا أكثر . سوف تدخل المجموعة الرئيسية السجلات ، ثم تتصل بالإجراءات المناسبة .

وفيما يلي محتويات البرنامج :

```
PROGRAM billing1(input,output);

(* THIS PROGRAM ILLUSTRATES THE USE OF RECORDS
   IN A SIMPLE CUSTOMER BILLING SYSTEM *)

TYPE status = (current,overdue,delinquent);
line = PACKED ARRAY [1..30] OF char;
date = RECORD
    month : 1..12;
    day : 1..31;
    year : 1900..2100
END;
```

(تكملة البرنامج في الصفحة التالية)

```

account = RECORD
    name : line;
    street : line;
    city : line;
    custno : 1..9999;
    custtype : status;
    oldbalance : real;
    newbalance : real;
    payment : real;
    paydate : date
END;
VAR customer : ARRAY [1..100] OF account;
    i,n : 1..100;

PROCEDURE readinput;
(* Read input data for each record *)
VAR count : 1..30;
    slash : char;
BEGIN
    FOR i := 1 TO n DO
        WITH customer [i] DO
            BEGIN
                writeln;
                writeln('Customer no. ',i:3);
                write('   Name: ');
                count := 1;
                REPEAT
                    read(name [count]);
                    count := count + 1
                UNTIL eoln;
                readln;
                write('   Street: ');
                count := 1;
                REPEAT
                    read(street [count]);
                    count := count + 1
                UNTIL eoln;
                readln;
                write('   City: ');
                count := 1;
                REPEAT
                    read(city [count]);
                    count := count + 1
                UNTIL eoln;
                readln;
                write('   Account number: ');
                readln(custno);
                write('   Previous balance: ');
                readln(oldbalance);
                write('   Current payment: ');
                readln(payment);
                write('   Payment date (mm/dd/yyyy): ');
                WITH paydate DO
                    read(month,slash,day,slash,year);
                readln
            END
        END
    END;
END; (* readinput *)

```

(تكملة البرنامج في الصفحة التالية)

```

PROCEDURE processdata;
(* Determine status and calculate a new balance for each record *)
BEGIN
  FOR i := 1 TO n DO
    WITH customer [i] DO
      BEGIN
        custtype := current;
        IF (payment > 0) AND (payment < 0.1*oldbalance)
          THEN custtype := overdue;
        IF (oldbalance > 0) AND (payment = 0)
          THEN custtype := delinquent;
        newbalance := oldbalance - payment;
      END
    END;
  (* processdata *)

PROCEDURE writeoutput;
(* Write out current information for each record *)
BEGIN
  FOR i := 1 TO n DO
    WITH customer [i] DO
      BEGIN
        writeln;
        write('Name: ',name);
        writeln(' Account number: ',custno:4);
        writeln('Street: ',street);
        writeln('City: ',city);
        writeln;
        write('Old balance: ',oldbalance:7:2);
        write(' Current payment: ',payment:7:2);
        writeln(' New balance: ',newbalance:7:2);
        writeln;
        write('Account status: ');
        CASE custtype OF
          current   : writeln('CURRENT');
          overdue   : writeln('OVERDUE');
          delinquent : writeln('DELINQUENT')
        END;
        writeln;
      END
    END;
  (* writeoutput *)

BEGIN (* main action statements *)
  writeln('CUSTOMER BILLING SYSTEM');
  writeln;
  write('How many customers are there? ');
  readln(n);
  readinput;
  processdata;
  writeoutput;
END.

```

لاحظ أن مكون WITH موجود في كل إجراء من الإجراءات . وفي الواقع يحتوى readinput على مكون WITH مزيج ( أى متداخل ) يعد العناصر الفردية للسجل المحصور paydate ، والموجود داخل كل سجل من سجلات العملاء .

افرض الآن أن البرنامج يستخدم في تشغيل أربعة سجلات لعملاء وهميين . وفيما يلي حوار المدخلات ، مع وضع خط تحت استجابة المستخدم ..

## CUSTOMER BILLING SYSTEM

How many customers are there? 4

Customer no. 1

Name: Richard L. Warren  
Street: 123 Vistaview Drive  
City: Denver, CO  
Account number: 4208  
Previous balance: 247.88  
Current payment: 25.00  
Payment date (mm/dd/yyyy): 6/14/1983

Customer no. 2

Name: Marcia Korenstein  
Street: 4383 Affluent Avenue  
City: Beechview, OH  
Account number: 2219  
Previous balance: 135.00  
Current payment: 135.00  
Payment date (mm/dd/yyyy): 8/10/1983

Customer no. 3

Name: Mark Singer  
Street: 1787 Larynx Lane  
City: Indianapolis, IN  
Account number: 8452  
Previous balance: 387.42  
Current payment: 35.00  
Payment date (mm/dd/yyyy): 7/4/1983

Customer no. 4

Name: Phyllis W. Smith  
Street: 1000 Great White Way  
City: New York, NY  
Account number: 711  
Previous balance: 260.00  
Current payment: 0  
Payment date (mm/dd/yyyy): 11/27/1983

عند ذلك ينتج البرنامج بيانات المخرجات التالية كاستجابة للمدخلات سالفة الذكر .

Name: Richard L. Warren Customer number: 4208  
Street: 123 Vistaview Drive  
City: Denver, CO

(التكلمة في الصفحة التالية)



Old balance: 247.88 Current payment: 25.00 New balance: 222.88

Account status: CURRENT

Name: Marcia Korenstein Customer number: 2219  
Street: 4383 Affluent Avenue  
City: Beechview, OH

Old balance: 135.00 Current payment: 135.00 New balance: 0.00

Account status: CURRENT

Name: Mark Singer Customer number: 8452  
Street: 1787 Larynx Lane  
City: Indianapolis, IN

Old balance: 387.42 Current payment: 35.00 New balance: 352.42

Account status: OVERDUE

Name: Phyllis W. Smith Customer number: 711  
Street: 1000 Great White Way  
City: New York, NY

Old balance: 260.00 Current payment: 0.00 New balance: 260.00

Account status: DELINQUENT

من وجهة النظر العملية ، فإن طبيعة هذا البرنامج غير واقعية بعض الشيء . وبصفة خاصة ... فهو يعطى شعورا بسيطا بإدخال محتويات سجل العميل وتنفيذ حسابات بسيطة ، ثم كتابة سجل مجدد ، دون حفظ نسخة دائمة للمعلومات الجديدة داخل الكمبيوتر ( أو فى أى تخزين ثانوى آخر ) .

وسوف نرى فى الفصل التالى كيف يمكن حفظ سجلات العملاء حفظا دائما فى ملفات بيانات تلغى الحاجة إلى إعادة إدخال البيانات عندما يكون هناك حاجة إلى تجديدها .

#### 4. VARIANT RECORDS

#### ٤ - سجلات متغيرة

حتى الآن لم نعتبر إلا سجلات تظل مكوناتها ثابتة أثناء تنفيذ البرنامج . من الممكن أيضا - على أية حال - أن نعرف سجل يمكن أن تتغير محتوياته ( أو جزء من محتوياته ) داخل البرنامج ، طبقا للقيمة التى تحدد لبعض الحقول الخاصة الموجودة داخل السجل ( حقل الشعار tag field ) مثل هذا النوع من السجلات يعرف بأنه سجل متغير Vari-ant record ( أو الجزء المتغير Variant part من السجل ) .

والصيغة العامة لتعريف نوع السجل تحقوى على ( كل من ) جزء ثابت ، وجزء متغير كما يلى :

TYPE record name = RECORD

fixed field 1;

fixed field 2;

(التكملة فى الصفحة التالية)

```

CASE tag field identifier : type OF
  case label 1 : variant fieldlist 1;
  case label 2 : variant fieldlist 2;
  .
  .
  .
END

```

كل قائمة حقول field list يعبر عنها في صورة تعداد الحقول ، أى :

(variant field 1, variant field 2, . . . , variant field n)

يجب أن يصاحب حقول الشعار نوع بيانات ترتيبى سبق تعريفه . ويجب أن تظهر كل قيمة لهذا النوع الترتيبى مرة واحدة فقط كعنوان حالة Case label ، وذلك فى جزء لاحق للمتغير فى تعريف السجل . ويحتوى على ذلك الجزء النشط من السجل على الحقول التى تناظر بعض عناوين الحالة الخاصة كما هو محدد لمعرفة حقول الشعار .

كما يمكن أن يوجد جزء متغير فقط فى أى سجل ، ويجب أن يتبع الجزء الثابت من السجل دائما .

مثال (١٠-٢٢)

فيما يلى مثالا لتعريف سجل يحتوى على كل من الجزء الثابت والجزء المتغير .

```

TYPE item = (stereo,tv);
name = PACKED ARRAY [1..80] OF char;
inventory = RECORD
  stockno : 1..20000;
  supplier : name;
  quantity : integer;
  CASE itemtype : item OF
    stereo : (power : 1..1000);
    tv : (tubesize : 1..25; color : char)
  END;
VAR stockitem,backorder : inventory;

```

ونوع السجل مسمى inventory . يحتوى الجزء الثابت منه على الحقول stockno , supplier , quantity ، ويحتوى الجزء المتغير على حقول power أو الحقول tubesize , color طبقا للقيمة التى تحدد لمعرفة حقول شعار هو item type . إذا ما حدد stereo بصفة خاصة لـ item type فعلى هذا يحتوى الجزء المتغير من السجل على الحقول power ، أما إذا ما حدد tv > item type فيحتوى - على هذا - الجزء المتغير على الحقول tubesize , color .

لاحظ أننا شملنا توضيحا لمتغيرين من نوع السجل ، وهما stockitem , backorder . كل من هذين المتغيرين يمثل سجلا من نفس نوع inventory .

ويمكن الاتصال بالعناصر الفردية للسجل المتغير وبمعرفة حقول شعار الطريقة المستخدمة مع العناصر الفردية

للسجل الثابت ، وذلك بكتابة مايلي :

*variable name.field name*

*variable name.field name.sub-field name*

أو كتابة مايلي :

*variable name.field name [index values]*

وذلك طبقا لنوع البيانات الخاص بالعنصر الفردي للسجل .

مثال (١٠-٢٢)

فيما يلي عدة عبارات غير مرتبطة ببعضها ، تستخدم عناصر فردية من سجل . وتشير كل العبارات إلى عناصر للمتغيرات من نوع السجل المعرفة في مثال (١٠ - ٢٢) .

```
stockitem.stockno := 12345;

backorder.quantity := 15;

stockitem.itemtype := stereo;

stockitem.power := 150;

writeln(backorder.stockno,backorder.power);

IF stockitem.quantity < 24 THEN
  BEGIN
    writeln(stockitem.supplier);
    writeln(stockitem.tubesize,stockitem.color)
  END;
```

لاحظ أن بعض العبارات تستخدم حقولا ثابتة ، بينما تستخدم عبارات أخرى حقولا متغيرة . وتحتوى العبارتان الأخيرتان على إشارات لحقول ثابتة وحقول متغيرة . لاحظ أيضا أنه تحددت قيمة لمعرف حقل شمار في العبارة الثالثة .

يبين مثال (١٠ - ٢٢) معالم معينة يجب أن يكون المبرمج حريصا من ناهيتها . أولا : يجب أن تكون أسماء الحقول داخل السجل ، بما فيها أسماء الحقول المتغيرة فردية . ( وفي بعض التطبيقات يوجد إغراء باستخدام نفس اسم الحقل في قائمتين أو أكثر للحقول المتغيرة ) . ثانيا : يجب تحديد قيمة مناسبة لمعرف حقل شعار إذا ما كان مطلوباً الاتصال بحقل متغير واحد . وعلى هذا ... إذا ما أريد الاتصال في مثال (١٠ - ٢٢) بـ *stockitem.power* ، فيجب أن يمثل *stockitem.itemtype* بأنه *stereo* .

يمكن استخدام مكون *WITH* مع سجلات من النوع المتغير ، مثل استخدامها تماما مع سجلات النوع الثابت ( انظر القسم ٣ من هذا الفصل ) . وهذا يعيل إلى تقليل احتمالات الخطأ ، وتحسين القراءة الشاملة للبرنامج .

### مثال (١٠-٢٤)

افترض أننا نريد تجديد حقولا محددة في سجل stockitem المعروف في مثال (١٠-٢٢) . يمكن تحقيقه بسهولة باستخدام مكوّن WITH كما يلي :

```
WITH stockitem DO
  BEGIN
    quantity := quantity - sales;
    IF quantity < 0 THEN backorder.quantity := -quantity;
    IF itemtype = stereo THEN writeln(power)
                                ELSE writeln(tubesize,color)
  END;
```

لاحظ أن هذا المثال يحتوى على استخدام حقل ثابت ( quantity ) ، وحقل شعار ( item type ) ، وحقل عديدة متغيرة ( color , tubesize , power ) . لاحظ أيضا الإشارة لحقل واحد داخل سجل آخر ( backorder ) وتكتب الإشارة إلى الحقل في صورتها الكاملة ( أى على هيئة backorder . quantity ) حيث إن عبارة WITH لاتس على هذا السجل .

وهناك العديد من الاختلافات في شكل السجل العام يمكن ذكرها . فمن الممكن - على سبيل المثال - تعرّ سجل لايحتوى على جزء ثابت . أكثر من هذا ... يمكن أن تكون قائمة الحقول المتغيرة فارغة إذا ماكانت هناك رغبة ذلك . ويحدد هذا بواسطة فئة فارغة من الأقواس ، يليها عنوان الحالة المناسب . وكل من هاتين الحالتين موضح المثال التالى :

### مثال (١٠-٢٥)

اعتبر تعبير السجل المتغير التالى :

```
TYPE status = (single,married,divorced,widowed);
background = RECORD
  CASE maritalstatus : status OF
    single : ();
    married : (children : 0..10);
    divorced,widowed : (children : 0..10;
                        remarried : boolean)
  END;
VAR employees : ARRAY [1..100] OF background;
```

في هذا المثال employees هي منظومة ذات بعد واحد ، وعناصرها سجلات من نفس نوع background . لاحظ أن تعريف هذا السجل الخاص لايحتوى على جزء ثابت . لاحظ أيضا أن أول حقل متغير list ( المناظر لعدد الحالة single ) فارغ .

كما نرى أيضا أن آخر قائمة حقل متغير تناظر عناوين الحالة المختلفين divorced , widowed . ولاحظ أن الحقل children مشترك مع كل من قائمتى الحقول المتغيرتين الثانية والثالثة ( أى مع عناوين الحالة orced widowed, married ) . وتقتّرح طبيعة هذا المثال أن الجزء الثابت من السجل يمكن أن يكون موجودا أيضا ، محاذ على اسم name وعنوان address ورقم الموظف employee number .. الخ . ويمكن ( للقارئ ) إضافة هذه الحيل ببساطة . وهناك موقف آخر يحتاج لمناقشة ، وهو وجود معرف حقل شعار داخل تعريف السجل . وتواجد معرف

شعار ليس ضروريا . فإذا لم يتواجد ، فتحدد الحقول المتغيرة النشطة ضمنيا بواسطة حقل متغير يمكن الاتصال به . وبصفة خاصة الإشارة إلى حقل متغير خاص تنشط عنوان حالة خاصة ، وبذلك تصبح كل الحقول المتغيرة المصاحبة لعنوان الحالة هذا نشطة . وتقترح البرمجة الجيدة - على أية حال - عدم استخدام هذه السمة بصفة عامة ، خاصة في حالة المبرمجين المبتدئين .

مثال (١٠-٢٦)

اعتبر التخطيط الهيكلي للبرنامج التالي :

```

TYPE item = (stereo,tv);
  name = PACKED ARRAY [1..80] OF char;
  inventory = RECORD
    stockno : 1..20000;
    supplier : name;
    quantity : integer;
    CASE item OF
      stereo : (power : 1..1000);
      tv : (tubesize : 1..25;
            color : char)
    END;
VAR stockitem,backorder : inventory;
BEGIN
  WITH stockitem DO
    BEGIN
      .
      .
      tubesize := 19;
      color := 'Y';
      .
      .
    END
  END.

```

لاحظ أن تعريف السجل يحتوى على نوع حقل شعار ( وهو item ) ، وليس على معرف حقل شعور . وعلى هذا ... فالإشارات إلى tubesize , color داخل مجموعة الإجراءات الرئيسية تشمل أن قائمة الحقول المتغير الثانية نشطة ( أى أنها تناظر عنوان الحالة tv ) .

وبالرغم من أن أحد السجلات يمكن أن يكون له جزء ثابت واحد ، وجزء متغير واحد فقط ، فمن الممكن أن يوجد تعريف متغير واحد داخل تعريف آخر . وعلى هذا ... فيمكننا إنتاج حقولا داخل سجل فردى .

مثال (١٠-٢٧)

فيما يلي صورة أخرى لتعريف السجل الموجود في مثال (٢٢ - ١٠) . (أنظر الصفحة التالية)

```

TYPE item = (stereo,tv);
  style = (portable,stationary);
  name = PACKED ARRAY [1..80] OF char;
  inventory = RECORD
    stockno : 1..20000;
    supplier : name;
    quantity : integer;
    CASE itemtype : item OF
      stereo : (power : 1..1000);
      tv : (tubesize : 1..25;
        color : char;
        CASE size : style OF
          portable : (weight : 1..100;
            voltage : 1..220);
          stationary : ());
    END;
VAR stockitem,backorder : inventory;

```

لاحظ قائمة الحقول المتغيرة الثانية ( المناظرة لعنوان الحالة tv ) . تحتوى قائمة الحقول هذه على تعريف متغير آخر يعتمد على القيمة المحددة لمعرف حقل شعار ، وهو size . فإذا كان size يمثل portable . فإن قائمة الحقول المتغير tv تحتوى على الحقلين tubesize ، color فقط .

يجب أن يلاحظ القارئ أن التعريف المتغير الداخلى جدا له حقل فردى ( size ) يصاحب نوع بياناته المتعدد ( style ) . لاحظ أيضا قائمة الحقول الفارغة المناظرة لعنوان الحالة المتداخل stationary . وأخيرا عليك أن تلاحظ أن الحقول المتغيرة المصاحبة لـ tv تتبع الحقول الثابتة كما هو مطلوب .

ويجب أن يكون مفهوما أن مجموعة جديدة من الحقول المتغيرة سوف تصبح نشطة عندما يتم تحديد قيمة جديدة لمعرف حقل شعار . أكثر من هذا ... إذا لم يتواجد معرف شعار ، فسوف تصبح مجموعة جديدة من الحقول المتغيرة variants نشطة كنتيجة للإشارة إلى حقل متغير غير نشط . أى تغير غير مطلوب فى الحقول المتغيرة النشطة يجب تجنبه على ذلك . ويجب أن يكون المبرمج حريصا من ناحية هذه المشكلة فى البرامج التى تستخدم إجراءات أو دوال .

ولكى يقل احتمال مثل هذه الأخطاء ، لايسمح بتغيير حقل متغير variant تحت الظروف التالية :

١ - داخل مكون WITH إذا ظهر اسم الحقل المتغير فى عبارة WITH .

٢ - عندما يمر الحقل المتغير إلى إجراء أو دالة ، كمؤشر متغير فعلى .

أكثر من ذلك ... لايمكن أن يمر معرف حقل شعار إلى إجراء أو دالة كمؤشر متغير فعلى .

مثال (١٠-٢٨)

مراقبة المخزون . دعنا نطور الآن نظام مراقبة مخزون بسيطاً يشبه نظام فواتير العملاء الموضح فى مثال (١٠-٢١) . وسوف نستخدم الآن سجلات تحتوى على كل من الجزء الثابت والجزء المتغير . وبصفة خاصة اعتبر نظاماً يتتبع نوعية من السلع ، وهى أجهزة ستريو stereos ، وأجهزة التلفزيون television . يتم ادخال المعلومات التالية دوريا داخل الكمبيوتر : رقم التخزين stock number ( وهو رقم غير سالب ) ، ونوع العنصر item type ( "s" إذا كان جهاز ستريو ، و "t" لجهاز التلفزيون ) واسم المورد supplier's name ، ومسئول المخزون الأسمى ( رقم غير سالب ) ، والتغيير الحالى فى مسئول المخزون ( رقم سالب أو موجب أو صفر ) .

ذلك بالإضافة إلى بعض المعلومات الوصفية التي تضاف إلى كل عنصر ، وتكون طبيعتها معتمدة على نوع العنصر نفسه .

فبالنسبة لجهاز ستريو ، يتم إدخال مستوى الطاقة ( رقم صحيح يتراوح من ١ إلى ١٠٠٠ ) . وبالنسبة لجهاز التلفزيون ، يكون هناك تحديد إذا ما كان الجهاز قابلاً للنقل "y" أم لا "n" وكذلك حجم الشاشة ( رقم صحيح يتراوح من ١ إلى ٢٥ ) وتحديد ما إذا كان الجهاز ملوناً "y" أو لا "n" ، وكذلك إذا كان الجهاز قابلاً للنقل ، فيحدد الوزن ( من ١ إلى ١٠٠ ) والفولت ( من ١ إلى ٢٢٠ ) يتم إدخالها أيضا .

وتوضيح السجل المناسب موضح كما يلي :

```
TYPE item = (stereo,tv);
  style = (portable,stationary);
  name = PACKED ARRAY [1..80] OF char;
  inventory = RECORD
    stockno : 0..20000;
    supplier : name;
    quantity : integer;
    CASE itemtype : item OF
      stereo : (power : 1..1000);
      tv : (tubesize : 1..25;
        color : char;
        CASE size : style OF
          portable : (weight : 1..100;
            voltage : 1..220);
          stationary : ());
    END;
  VAR stockitem,backorder : ARRAY [1..100] OF inventory;
```

لاحظ أن سجل inventory يحتوى على أجزاء متغيرة متداخلة . لاحظ أيضا أننا عرفنا منظومتين مختلفتين ، هما stockitem ، backorder تحتويان على سجلات من نفس نوع inventory . يحتوى stockitem على سجلات مخزون كما تم إدخالها في الكمبيوتر ، ويحتوى backorder على معلومات عن العناصر التي يحدث لها أوامر خلفية فقط ( أى العناصر التي بها عجز ) .

وتحتوى العملية الكاملة على الخطوتين التاليتين :

١ - أقرأ كل بيانات المدخلات لكل سجل ، واخضبب مستوى المخزون طبقا لذلك .

٢ - بعد قراءة كل السجلات داخل الكمبيوتر ، وتضبيط مستويات المخزون طبقا لها ، اكتب السجلات موضحا مستويات المخزون الجديدة .

وبدلا من تحديد عدد السجلات مقدما - كما فعلنا مع نظام الفواتير الموضح في مثال (٢١ - ١٠) - دعنا نستمر في إدخال سجلات جديدة حتى يتم إدخال قيمة صفر في رقم التخزين . وعلى هذا ... تبدو المجموعة الإجرائية الأساسية على النحو التالي :

```
BEGIN
  writeln('INVENTORY CONTROL SYSTEM');
  writeln;
  writeln('... DATA INPUT');
```

(أنظر الصفحة التالية)

```

writeln;
writeln(' When finished, enter 0 for the stock number');
writeln;
i := 0;
REPEAT
  i := succ(i);
  readinput;
UNTIL stockitem [i].stockno = 0;
n := pred(i);
IF n > 0 THEN writeoutput
END.

```

لاحظ أن  $i$  هي عدد سجلات من النوع العددي الصحيح تتراوح قيمته من 0 إلى  $n$  ، حيث  $n$  هي إجمالي عدد السجلات التي يتم إدخالها . ( وتحدد آخر قيمة  $i$  بأنها  $n$  ) ، كما أن `readinput` , `writeoutput` إجراءان ينتج عنهما قراءة بيانات المدخلات داخل الكمبيوتر ، وتضبيب مستوى المخزون ، وكتابة البيانات الجديدة خارج الكمبيوتر على التوالي .

والإجراء `read input` متداخل ، وبه عدد من الملقنات ( التقليدية ) المناسبة ، والبرمجة مباشرة ، بالرغم من أنها طويلة بعض الشيء . وعلى هذا يكون لدينا مايلي :

```

PROCEDURE readinput;
(* Enter input data for one record *)
VAR count : 1..80;
BEGIN
  WITH stockitem [i] DO
    BEGIN
      writeln;
      write('Stock no.: ');
      readln(stockno);
      IF stockno > 0 THEN
        BEGIN
          write('Item type (s/t): ');
          readln(designator);
          IF (designator = 's') OR (designator = 'S')
            THEN BEGIN (* stereo *)
              itemtype := stereo;
              write('Power level: ');
              readln(power)
            END
          ELSE BEGIN (* tv *)
              itemtype := tv;
              write('Portable TV? (y/n) ');
              readln(designator);
              IF (designator = 'y') OR (designator = 'Y')
                THEN BEGIN
                  size := portable;
                  write('Weight: ');
                  readln(weight);
                  write('Voltage: ');
                  readln(voltage)
                END
            END
        END
      END
    END
  END

```

(التكملة في الصفحة التالية)



```

        ELSE size := stationary;
        write('Tube size: ');
        readln(tubesize);
        write('Color: (y/n) ');
        readln(color)
    END;
    write('Supplier: ');
    count := 1;
    REPEAT
        read(supplier [count]);
        count := succ(count)
    UNTIL eoln;
    write('Original inventory level: ');
    readln(quantity);
    write('Change in inventory level: ');
    readln(change);
    quantity := quantity + change;
    IF quantity < 0 THEN
        BEGIN
            backorder [i] := stockitem [i];
            backorder [i].quantity := abs(quantity);
            quantity := 0
        END
    END (* IF stockno > 0 *)
    END (* WITH stockitem [i] *)
END; (* readinput *)

```

المحدد في هذا الاجراء متغير شامل من النوع الحرفي ، يعرف نوع المدخلات . كما أن change متغير شامل من النوع الصحيح ، يمثل التغير في مستوى المخزون . لاحظ أن change يمكن أن يأخذ قيما موجبة أو سالبة طبقا لما إذا كان هناك سحب من المخزون ، أو إعادة ملء المخزون . لاحظ أيضا أن هذا الإجراء مكتوب للحصول على سجل واحد في نفس الوقت ( حيث إنه غير معروف مسبقا عدد السجلات التي سبق إدخالها في الكمبيوتر ) .

وهناك سمة ملحوظة أخرى خاصة بإجراء readinput ، وهي حقيقة أن تجديد المخزون يحدث بمجرد إدخال البيانات الجديدة . فإذا كان مستوى المخزون الجديد سالبا ، فتنقل المعلومات من stockitem [i] إلى backorder [i] ويخزن العجز كقيمة موجبة . واستخدام backorder [i] ليس ضروريا ، ويضيف القليل في واقع الأمر إلى البرنامج ، فيما عدا أنه يقترح كيف يمكن حفظ مثل هذا الانزواج من السجلات المتناظرة في برنامج أكثر واقعية ( المزيد عن ذلك يذكر فيما بعد ) .

دعنا نعتبر الآن الإجراء writeoutput . هذا الإجراء ليس في حاجة لأن يكون متداخلا ، ويمكنه تشغيل كل السجلات في وقت واحد . وعلى هذا يمكننا كتابته كما يلي :

```

PROCEDURE writeoutput;
(* Write output data for all records *)
BEGIN
    writeln('INVENTORY CONTROL SYSTEM');
    writeln;
    writeln('    CURRENT INVENTORY');
    writeln;
    FOR i := 1 TO n DO
        WITH stockitem [i] DO

```

(التكملة في الصفحة التالية)

```

BEGIN
  writeln;
  writeln('Stock no.: ',stockno:5);
  CASE itemtype OF
    stereo : writeln('Stereo: ',power:4,' watts');
    tv : BEGIN
      IF size = portable
        THEN write('Portable ');
      IF (color = 'y') OR (color = 'Y')
        THEN write('Color TV')
        ELSE write('B & W TV');
      writeln(' ',tubesize:2,'-inch tube');
      IF size = portable THEN
        BEGIN
          write('Weight: ',weight:3,' pounds ');
          writeln('Voltage: ',voltage:3,' volts')
        END
      END (* tv *)
    END; (* CASE itemtype *)
  writeln('Supplier: ',supplier);
  IF quantity > 0
    THEN writeln('Present inventory: ',quantity:3)
    ELSE writeln('Quantity backordered: ', backorder [i].quantity:3);
  writeln
END (* WITH stockitem [i] *)
END; (* writeoutput *)

```

لاحظ أن أحد حقول backorder [i] يكتب في حالة ما إذا كان هناك عجز في المخزون . وفيما يلي محتويات برنامج البسكال كاملة .

```

PROGRAM inventory1(input,output);
(* THIS PROGRAM ILLUSTRATES THE USE OF VARIANT RECORDS
   IN A SIMPLE INVENTORY CONTROL SYSTEM *)
TYPE item = (stereo,tv);
  style = (portable,stationary);
  name = PACKED ARRAY [1..80] OF char;
  inventory = RECORD
    stockno : 0..20000;
    supplier : name;
    quantity : integer;
    CASE itemtype : item OF
      stereo : (power : 1..1000);
      tv : (tubesize : 1..25;
        color : char;
        CASE size : style OF
          portable : (weight : 1..100;
            voltage : 1..220);
          stationary : ());
    END;
  END;

```

(تكملة البرنامج في الصفحة التالية)

```

VAR stockitem,backorder : ARRAY [1..100] OF inventory;
    designator : char;
    i,n,change : integer;

PROCEDURE readinput;
(* Read input data for one record *)
VAR count : 1..80;
BEGIN
    WITH stockitem [i] DO
        BEGIN
            writeln;
            write('Stock no.: ');
            readln(stockno);
            IF stockno > 0 THEN
                BEGIN
                    write('Item type (s/t): ');
                    readln(designator);
                    IF (designator = 's') OR (designator = 'S')
                        THEN BEGIN (* stereo *)
                            itemtype := stereo;
                            write('Power level: ');
                            readln(power)
                        END
                    ELSE BEGIN (* tv *)
                            itemtype := tv;
                            write('Portable TV? (y/n) ');
                            readln(designator);
                            IF (designator = 'y') OR (designator = 'Y')
                                THEN BEGIN
                                    size := portable;
                                    write('Weight: ');
                                    readln(weight);
                                    write('Voltage: ');
                                    readln(voltage)
                                END
                            ELSE size := stationary;
                            write('Tube size: ');
                            readln(tubesize);
                            write('Color: (y/n) ');
                            readln(color)
                        END;
                    write('Supplier: ');
                    count := 1;
                    REPEAT
                        read(supplier [count]);
                        count := succ(count)
                    UNTIL eoln;
                    write('Original inventory level: ');
                    readln(quantity);
                    write('Change in inventory level: ');
                    readln(change);
                    quantity := quantity + change;
                    IF quantity < 0 THEN
                        BEGIN
                            backorder [i] := stockitem [i];
                            backorder [i].quantity := abs(quantity);
                            quantity := 0
                        END
                END
            END;
        END;
    END;

```

(تكملة البرنامج في الصفحة التالية)

```

        END  (* IF stockno > 0 *)
    END  (* WITH stockitem [i] *)
END:  (* readinput *)

PROCEDURE writeoutput;
(* Write output data for all records *)
BEGIN
    writeln('INVENTORY CONTROL SYSTEM');
    writeln;
    writeln('    CURRENT INVENTORY');
    writeln;
    FOR i := 1 TO n DO
        WITH stockitem [i] DO
            BEGIN
                writeln;
                writeln('Stock no.: ',stockno:5);
                CASE itemtype OF
                    stereo : writeln('Stereo: ',power:4,' watts');
                    tv : BEGIN
                        IF size = portable
                            THEN write('Portable ');
                        IF (color = 'y') OR (color = 'Y')
                            THEN write('Color TV')
                            ELSE write('B & W TV');
                        writeln(' ',tubesize:2,'-inch tube');
                        IF size = portable THEN
                            BEGIN
                                write('Weight: ',weight:3,' pounds ');
                                writeln('Voltage: ',voltage:3,' volts')
                            END
                        END
                    END  (* tv *)
                END;  (* CASE itemtype *)
                writeln('Supplier: ',supplier);
                IF quantity > 0
                    THEN writeln('Present inventory: ',quantity:3)
                    ELSE writeln('Quantity backordered: ',backorder [i].quantity:3);
                writeln
            END  (* WITH stockitem [i] *)
        END;  (* writeoutput *)
    END;

BEGIN  (* main action statements *)
    writeln('INVENTORY CONTROL SYSTEM');
    writeln;
    writeln('    DATA INPUT');
    writeln;
    writeln(' When finished, enter 0 for the stock number');
    writeln;
    i := 0;
    (* read input records *)
    REPEAT
        i := succ(i);
        readinput;
    UNTIL stockitem [i].stockno = 0;
    n := pred(i);
    (* write output records *)
    IF n > 0 THEN writeoutput
END.
```

افرض أن البرنامج استخدم الآن لتشغيل ثلاثة سجلات توضيحية . يبدو حوار المدخلات على النحو التالي :

# INVENTORY CONTROL SYSTEM

## DATA INPUT

When finished, enter 0 for the stock number

Stock no.: 12417  
Item type (s/t): s  
Power level: 150  
Supplier: House of Audio  
Original inventory level: 200  
Change in inventory level: -12

Stock no.: 912  
Item type (s/t): t  
Portable TV? (y/n) n  
Tube size: 23  
Colour: (y/n) y  
Supplier: Ace Radio & TV  
Original inventory level: 8  
Change in inventory level: 72

Stock no.: 644  
Item type (s/t): t  
Portable TV? (y/n) y  
Weight: 25  
Voltage: 110  
Tube size: 12  
Color: (y/n) n  
Supplier: Crazy Harry's  
Original inventory level: 12  
Change in inventory level: -15

Stock no.: 0

في هذا الحوار يقدم الكمبيوتر ملقنات ، ويقوم المستفيد بإدخال المعلومات المطلوبة ( استجابات المستفيد موضوع تحتها خط ) .

لاحظ أن حوار المدخلات يستمر حتى تدخل قيمة لرقم المخزون stock number مساوية صفرًا .

وينتج البرنامج المخرجات التالية كاستجابة لبيانات المدخلات :

## INVENTORY CONTROL SYSTEM

### CURRENT INVENTORY

Stock no.: 12417  
Stereo: 150 watts  
Supplier: House of Audio  
Present inventory: 188  
  
Stock no.: 912  
Color TV 23-inch tube  
Supplier: Ace Radio & TV  
Present inventory: 80  
  
Stock no.: 644  
Portable B & W TV 12-inch tube  
Weight: 25 pounds Voltage: 110 volts  
Supplier: Crazy Harry's  
Quantity backordered:

وعلى القارئ أن يتذكر مرة أخرى أن هذا البرنامج - مثل برنامج فواتير العملاء المقدم في مثال (١٠ - ٢١) - مبسط جدا ، حيث إنه لا يحفظ سجلات المخزون دائما داخل ملف بيانات . وفي واقع الأمر ... فإن كل ملفات نظم مراقبة المخزون تحتوى على مثل هذه الإمكانيات للتخزين الدائم . ويعد ذلك يمكن استرجاع السجلات الفردية من الملف ، وإجراء التعديل عليها ، وكتابة محتوياتها ، ثم تخزينها مرة أخرى . وسوف نرى كيف يمكن تحقيق ذلك في الفصل القادم . ويجب أن يكون واضحا أن المثال الحالي يهدف إلى توضيح كيفية تشغيل بيانات من نوع السجلات في البسكال فقط .

## Review Questions

## أسئلة للمراجعة

- (١) ماهو الفرق الأساسى بين السجل والمنظومة ؟
- (٢) ماذا يعنى الحقل ؟ وماهى العلاقة بين الحقل والسجل ؟
- (٣) لخص قواعد تعريف السجل ؟
- (٤) لخص قواعد تعريف متغير من نوع السجل . قارن ذلك بإجابتك للسؤال السابق .
- (٥) ماهو نوع البيانات الذى يمكن أن يصاحب عنصر بيانات فردياً داخل الحقل ؟
- (٦) فى أى نوع من أنواع التطبيقات يكون من المرغوب فيه لعنصر سجل أن يصبح منظومة ؟ وضح ذلك .
- (٧) فى أى نوع من أنواع التطبيقات يكون من المرغوب فيه لعنصر منظومة أن يكون سجلا ؟ وضح ذلك ، وقارن مع إجابتك للسؤال السابق .
- (٨) كيف يمكن ضغط سجل ؟ وماهى عيوب ومميزات استخدام السجلات المضغوطة ؟

- (٩) هل يمكن استخدام نفس اسم الحقل فى سجلين مختلفين ؟ وضح ذلك .
- (١٠) هل يمكن تحديد محتويات أحد السجلات لسجل آخر ؟ ماهى القيود التى تقع على هذا النوع من التحديد ؟
- (١١) ماهى طرق استخدام عناصر السجل داخل برنامج البسكال ؟
- (١٢) ماهو المحدد للحقل ؟ وكيف يكتب ؟ وكيف يستخدم ؟
- (١٣) افرض أن عنصر فردى من سجل عبارة عن عنصر بيانات مرتب . كيف يمكن الاتصال بعنصر واحد من عنصر البيانات هذا ؟
- (١٤) ماهو الغرض من مكون WITH ؟
- (١٥) لخص قواعد استخدام مكون WITH . هل يمكن استخدام أسماء سجلات متعددة فى مكون WITH واحد ؟
- (١٦) افرض أن عبارة WITH واحدة تحتوى على متغيرات متعددة من نوع السجل ، وبها اسم حقل مشترك . أى سجل من هذه السجلات يتم الاتصال به عندما يتم الاتصال بحقل داخل المكون ؟ وكيف يمكن الاتصال بحقل له نفس الاسم ، لكنه ينتمى إلى سجل آخر ؟
- (١٧) اذكر عدة مميزات لاستخدام مكون WITH عندما يكون ذلك مناسباً .
- (١٨) ماذا يعنى الجزء المتغير من السجل ؟ وكيف يختلف هذا الجزء عن الجزء الثابت ؟
- (١٩) هل يجب أن يوجد فى كل سجل جزء متغير ؟ وهل يجب أن يوجد به جزء ثابت ؟ وهل يمكن أن يكون فى السجل أكثر من جزء متغير واحد ؟
- (٢٠) لخص قواعد تعريف سجل به كل من الجزء الثابت والجزء المتغير . ضاه تعريف الجزء المتغير مع مكون الحالة CASE الذى سبق ذكره فى الفصل السادس .
- (٢١) إذا ما احتوى السجل على جزء متغير ، فأين يجب أن يقع هذا الجزء ؟
- (٢٢) ماهو حقل شعار ؟ وماهو الغرض منه ؟
- (٢٣) مانوع البيانات التى يمكن أن تصاحب حقل شعار ؟
- (٢٤) هل يجب أن يوجد حقل شعار فى كل سجل متغير ؟ وضح ذلك .
- (٢٥) كيف يمكن الاتصال بعناصر فردية موجودة داخل الجزء المتغير من السجل ؟ وماهى القيود الموضوعة على الاتصال بمثل هذه الحقول المتغيرة ؟
- (٢٦) هل يمكن استخدام مكون WITH مع سجلات من النوع المتغير ؟
- (٢٧) كيف يمكن تعريف قائمة حقل متغير فارغة ؟ ولماذا يمكن أن يكون هذا شيئاً مرغوباً فيه ؟

(٢٨) كيف يمكن عمل تداخل للأجزاء المتغيرة داخل سجل واحد ؟

(٢٩) لماذا لايسمح بالتغييرات فى الجزء المتغير داخل مكون WITH إذا ماظهر اسم الحقل المتغير فى عبارة WITH ؟

(٣٠) ماهى القيود التى يجب أخذها فى الاعتبار عندما يمرر حقل متغير إلى إجراء أو دالة كمؤشر متغير فعلى ؟

(٣١) هل يمكن تمرير معرف حقل شعار إلى إجراء أو دالة كمؤشر متغير فعلى ؟ وضح ذلك .

(٣٢) علق على استخدام الأجزاء المتغيرة بواسطة المبرمجين المبتدئين .

## solved Problems

## مسائل محلولة :

(٣٣) فيما يلى عدة توضيحات سجلات

```
(a) VAR sample : RECORD
      first  : 1..100;
      second : real;
      third  : boolean
    END;

(b) TYPE demo = RECORD
      first  : 1..100;
      second : real;
      third  : boolean
    END;
    VAR sample1,sample2 : demo;

(c) TYPE line = PACKED ARRAY [1..80] OF char;
      personal = RECORD
        name,street,city : line
      END;
    VAR employee : ARRAY [1..500] OF personal;

(d) TYPE color = (red,green,blue);
      sample = PACKED RECORD
        first : color;
        second : 1..132;
        third : char
      END;
    VAR trial : sample;

(e) TYPE color = (red,green,blue);
      sample = RECORD
        CASE select : color OF
          red   : (first : 100..199;
                   second : 200..299);
          green : (third : 300..399);
          blue  : ();
        END;
    VAR demo : ARRAY [1..500] OF sample;
```



لاحظ أن sample لا يحتوى على جزء ثابت . لاحظ أيضا أن قائمة الحقول المتغير الفارغ تتناظر اسم الحالة blue .

```
(f) TYPE color = (red,green,blue);
      texture = (coarse,fine);
      sample = PACKED RECORD
          first : color;
          second : 1..132;
          CASE color OF
              red : (first : 100..199;
                     second : 200..299);
              green : ();
              blue : (CASE texture OF
                      coarse : (third : 300..399);
                      fine : ());
          END;
VAR demo1,demo2 : ARRAY [1..99] OF sample;
```

لاحظ أن الأجزاء المتغيرة المتداخلة في sample لا تحتوى على معرفات حقول شعاع .

(٣٤) فيما يلي تخطيطات هيكلية لاستخدام سجلات وعناصر سجلات في مواقف برمجة تقليدية .

```
(a) PROGRAM sample(input,output);
      TYPE line = PACKED ARRAY [1..80] OF char;
          personal = RECORD
              name,street,city : line
          END;
      VAR employee : personal;
      BEGIN
          .
          .
          (* enter input data *)
          .
          .
          writeln(employee.name);
          writeln(employee.street);
          writeln(employee.city);
          .
          .
      END;

(b) PROGRAM sample(input,output);
      TYPE line = PACKED ARRAY [1..80] OF char;
          personal = RECORD
              name,street,city : line
          END;
      VAR employee : ARRAY [1..500] OF personal;
          count : 1..500;
      BEGIN
          .
          .
          (* enter input data *)
          .
          .
          FOR count := 1 TO 500 DO
              BEGIN
                  writeln(employee [count].name);
                  writeln(employee [count].street);
                  writeln(employee [count].city)
              END;
          .
          .
      END.
```

```
(c) PROGRAM sample(input,output);
    TYPE line = PACKED ARRAY [1..80] OF char;
        personal = RECORD
            name,street,city : line
        END;
    VAR employee : ARRAY [1..500] OF personal;
        count,total : 1..500;
    BEGIN
        .
        .
        (* enter input data *)
        .
        .
        FOR count := 1 TO total DO
            WITH employee [count] DO
                BEGIN
                    writeln(employee [count].name);
                    writeln(employee [count].street);
                    writeln(employee [count].city)
                END;
            .
            .
        END.

(d) PROGRAM sample(input,output);
    TYPE line = PACKED ARRAY [1..80] OF char;
        personal = RECORD
            name,street,city : line
        END;
    VAR customer,employee : ARRAY [1..500] OF personal
        i,j,total : 1..500;
    BEGIN
        .
        .
        (* enter input data *)
        .
        .
        FOR i := 1 TO total DO
            WITH customer [i] DO
                FOR j := 1 TO total DO
                    IF name = employee [j].name THEN
                        BEGIN
                            writeln(name);
                            writeln(street);
                            writeln(city)
                        END;
                    .
                    .
                END.
            .
            .
        END.
```

لاحظ أن عناصر السجل المكتوبة هي جزء في [i] customer ، وليست جزءا في [i] employee .

```
(e) PROGRAM sample(input,output);
TYPE status = (single,married,divorced,widowed);
   line = PACKED ARRAY [1..80] OF char;
   background = RECORD
       name,address : line;
       employeeno : 1..9999;
       CASE maritalstatus : status OF
           single : ();
           married : (children : 0..10);
           divorced,widowed :
               (children : 0..10;
                remarried : boolean)
       END;
VAR employees : ARRAY [1..100] OF background;
    count,total : 1..100;
BEGIN
    .
    .
    (* enter input data *)
    .
    .
    FOR count := 1 TO total DO
        WITH employees [count] DO
            BEGIN
                writeln(name);
                writeln(address);
                writeln(employeeno:4);
                IF maritalstatus <> single THEN writeln(children:2);
                IF ((maritalstatus = divorced) OR
                    (maritalstatus = widowed)) AND
                    (remarried = true) THEN maritalstatus = married
            END;
        .
        .
    END.
```

## Supplementary Problems

## مشاكل متكاملة :

(٣٥) التخطيطات الهيكلية التالية توضح مواقف عديدة مختلفة لاستخدام سجلات وعناصر سجلات ، وبعضها مكتوب بطريقة خاطئة ، عرف كل الأخطاء .

```
(a) CONST c1 = 12;
      c2 = 31;
      c3 = 2000;
      VAR date : RECORD
          month : 1..c1;
          day : 1..c2;
          year : 0..c3
      END;
```

```
(b) TYPE months = (jan,feb,mar,apr,may,jun,
                    jul,aug,sep,oct,nov,dec);
    days = (sun,mon,tue,wed,thu,fri,sat);
    date = RECORD
        name : days;
        month : months;
        day : 1..31;
        year : 0..maxint
    END;
VAR yesterday,today : date;
```

```
(c) TYPE item = RECORD
    sales : real;
    cost : real;
    profit : real;
    bonus : boolean
VAR list : ARRAY [1..1000] OF sales;
```

```
(d) TYPE line = PACKED ARRAY [1..80] OF char;
    nameandaddress = RECORD
        name,street,city : line
    END;
    personal = RECORD
        description : nameandaddress;
        custno : 1..9999;
        status : char;
        balance : real
    END;
VAR customer : ARRAY [1..10000] OF personal;
```

```
(e) PROGRAM sample(input,output);
    TYPE date = RECORD
        month : 1..12;
        day : 1..31;
        year : 1900..2100
    END;
    VAR birthday,today : date;
        count : 0..maxint;
    BEGIN
        .
        .
        WITH today DO readln(month,day,year);
        .
        .
        count := 0;
        REPEAT
            WITH birthday DO
                readln(month,day,year);
            IF birthday = today THEN count := succ(count);
        UNTIL year = 0;
        writeln(count);
        .
        .
    END.
```

```

(f) PROGRAM sample(input,output);
    TYPE line = PACKED ARRAY [1..80] OF char;
    date = PACKED RECORD
        month : 1..12;
        day : 1..31;
        year : 1900..2100
    END;
    VAR total,count : 1..maxint;
    today : date;
    birthday : RECORD
        name : line;
        birthdate : date
    END;

    BEGIN
        .
        .
        readln(total);
        WITH today DO readln(month,day,year);
        .
        .
        FOR count := 1 TO total DO
            WITH birthday DO
                BEGIN
                    readln(name);
                    birthdate := today
                END;
            .
            .
        END.

(g) TYPE item = (stereo,tv);
    name = PACKED ARRAY [1..80] OF char;
    inventory = RECORD
        stockno : 1..20000;
        supplier : name;
        CASE item OF
            stereo : (power : 1..1000);
            tv : (tubesize : 1..25;color : char);
        quantity : integer
    END;
    VAR stockitem,backorder : inventory;

(h) PROGRAM sample(input,output);
    TYPE status = (single,married,divorced,widowed);
    background = RECORD
        CASE maritalstatus : status OF
            single : ();
            married : (children : 0..10);
            divorced,widowed :
                (children : 0..10; remarried : boolean)
        END;
    VAR employees : ARRAY [1..100] OF background;
    BEGIN
        .
        .
        maritalstatus := single;
        writeln(divorced.children);
        .
        .
    END.

```

```
(i) PROGRAM sample(input,output);
    TYPE status = (single,married,divorced,widowed);
    background = RECORD
        CASE maritalstatus : status OF
            single : ();
            married : (children : 0..10);
            divorced,widowed :
                (children : 0..10;remarried : boolean)
        END;
    VAR employees : ARRAY [1..100] OF background;
        i : 1..100;

    PROCEDURE setup(VAR test : status);
    BEGIN
        .
        .
        IF test = widowed THEN test := single
            ELSE test := succ(test)
        .
        .
    END;

    BEGIN (* main action statements *)
        FOR i := 1 TO 100 DO
            WITH employees [i] DO
                BEGIN
                    .
                    .
                    setup(maritalstatus);
                    .
                    .
                END
            END
        END.
```

## Programming Problems

## مشاكل برمجة :

(٣٦) عدل برنامج فواتير العملاء الموجود في مثال (١٠ - ٢١) ، بحيث يمكن كتابة أى تقرير من التقارير التالية :

١ - تقرير حالة كل العملاء ( ينتجه البرنامج نفسه ) .

ب - تقرير حالة العملاء أصحاب الحسابات متأخر ومقصر فقط .

ج - تقرير حالة العملاء أصحاب الحسابات المقصرة فقط .

حدد جزءاً لانتاج قائمة ، عند تنفيذ البرنامج يستطيع أن يختار المستخدم منها التقرير المطلوب إنتاجه . وبعد ذلك يعود البرنامج إلى القائمة بعد طباعة التقارير ، وتكون هناك إمكانية لإنتاج عدة تقارير مختلفة .

(٣٧) عدل برنامج مراقبة المخزون الموجود في مثال (١٠ - ٢٨) ، بحيث يمكن كتابة أى تقرير من التقارير التالية :

أ - تقرير بقائمة كاملة للمخزون ( ينتجه البرنامج نفسه ) .

ب - تقرير بقائمة لكل العناصر التى لها أوامر خلفية .

ج - تقرير بقائمة لكل أجهزة سترير الموجودة حالياً في المخزن .

د - تقرير بقائمة لكل أجهزة التلفزيون الملون الموجودة حالياً في المخزن .

هـ - تقرير بقائمة كل أجهزة التلفزيون الأبيض والأسود الموجودة حالياً في المخزن .

و - تقرير بقائمة لكل الأجهزة القابلة للنقل الموجودة حالياً في المخزن .

ابدأ بإنتاج قائمة عندما ينفذ البرنامج ، مع السماح للمستخدم بأن يختار تقريراً معيناً ، وبعد ذلك يعود البرنامج إلى هذه القائمة بعد إنتاج كل تقرير ، وتكون هناك إمكانية لإنتاج التقارير المتنوعة .

(٣٨) أعد كتابة كل برنامج من برامج البسكال التالية ، بحيث إنها تستخدم تكوين بيانات من نوع السجلات .

أ - برنامج الاستهلاك المقدم في مثالى ( ٦ - ١٧ ) ، ( ٧ - ١٢ ) .

ب - البرنامج المعطى في مثال ( ٨ - ٩ ) لحساب عدد الأيام التى تقع بين تاريخين محددين .

ج - البرنامج المستخدم في قراءة وكتابة قائمة بالأسماء والعناوين الموجود في مثال ( ٩ - ٧ ) .

(٣٩) عدل برنامج البسكال الموجود في مثال ( ٩ - ٧ ) ، بحيث يمكنه أن يقرأ عدة أسماء وعناوين مختلفة داخل الكمبيوتر ، ويعيد ترتيبها ترتيباً أبجدياً ، ثم يكتبها بترتيبها الجديد . حدد تكوين بيانات من نوع السجل داخل البرنامج .

(٤٠) عدل البرنامج الذى ينتج pig latin الموجود في مثال ( ٩ - ٢٧ ) ، بحيث إنه يقبل أسطر متعددة من النص . مثل كل سطر من أسطر النص بأنه سجل منفصل مع وجود ثلاثة حقول داخل كل سجل ، وهى .

أ - السطر الأصلى في النص .

ب - عدد الكلمات الموجودة في السطر .

ج - سطر النص المعدل ( أى السطر مكتوباً باستخدام pig latin ) .

( حدد الزيادات الموجودة في المشكلة رقم ( ٥٠ من الفصل التاسع ) . أدخل علامات التنقيط . والحروف الكبيرة ، وأصوات الحروف المزوجة )

(٤١) اكتب برنامجا كاملا بلغة البسكال لكل مشكلة من مشاكل البرمجة التالية ، مستخدما تكوينات بيانات من نوع السجل .

١ - مشكلة تحديد متوسط درجات الكلية الموجودة فى المشكلة رقم ( ٥٢ من الفصل التاسع ) الجزء (أ) .

ب - الصيغة الأخرى لمشكلة تحديد متوسط درجات الكلية الموجودة فى المشكلة رقم ( ٥٢ من الفصل التاسع ) الجزء (د) .

ج - المشكلة التى تجرى توافقا بين أسماء البلاد والعواصم المناظرة لها ( انظر المشكلة رقم ( ٥٦ من الفصل التاسع ) .

(٤٢) اكتب برنامجا كاملا بلغة البسكال ، يقبل المعلومات التالية لكل فريق من فرق البيسبول base ball و فرق كرة القدم .

١ - اسم الفريق ، بما فى ذلك المدينة .

٢ - عدد مرات الكسب .

٣ - عدد مرات الخسارة .

وبالنسبة لفرق البيسبول تضاف المعلومات التالية :

١ - عدد الضربات ( فى كل الموسم ) hits .

٢ - عدد الدورات runs .

٣ - عدد الأخطاء .

٤ - عدد المباريات extra - inning .

وبالمثل أضف لفريق كرة القدم المعلومات التالية :

١ - عدد مرات الإيقاف ties .

٢ - عدد مرات اللمس touchdowns .

٣ - عدد الأجوال field goals .

٤ - إجمالى المكسب ( على مدى الموسم ) total yards gined .

٥ - إجمالى الخسارة total yards given up to oppanents .

أدخل هذه المعلومات الخاصة بكل الفرق ، ثم أعد ترتيب الفرق ، واطبع قائمة بها طبقا لسجلات المكسب والخسارة ، مستخدما أسلوب إعادة الترتيب المستخدم فى مثال ( ٩ - ٨ ) .



---

تأكد من استخدام تكوين بيانات من نوع السجل ، تشمل الجزء الثابت والجزء المتغير من السجل . ويجب أن تحفظ هذه السجلات في منظومة .

اختبر البرنامج ، مستخدماً إحصاءات حديثة لهذه الغرض .



## الفصل الحادى عشر

### الملفات

### Files

نركز انتباهنا الآن على تكوين هام آخر للبيانات وهو الملف . الملف file مثل المنظومة عبارة عن تجميع لعناصر بيانات كلها من نفس النوع . وعلى عكس المنظومة ، يمكن تخزين الملف فى إحدى وحدات التخزين المساعد ( أى وحدة تشغيل شريط ، أو وحدة تشغيل قرص ) . وعلى هذا ... فيسمح لنا تكوين الملف بالتخزين الدائم للمعلومات والاتصال بهذه المعلومات عند الحاجة لها . ويتطلب العديد من التطبيقات الهامة هذه الإمكانية .

#### 1. PRELIMINARIES

#### ١ - أساسيات

بصفة عامة يوجد نوعان من الملفات يمكن إنتاجها بواسطة برنامج الكمبيوتر ، وكذلك الاتصال بهما ، وهما الملفات الدائمة والملفات المؤقتة . وتحفظ الملفات الدائمة permanent files فى إحدى وحدات تشغيل التخزين المساعد . ويمكن الاتصال بمحتويات هذه الملفات وتعديلها ، إما بواسطة البرنامج الذى أنتجها ، أو بواسطة برنامج آخر ، وذلك فى أى وقت . ويشار فى البسكال إلى الملفات الدائمة بأنها ملفات خارجية external files .

وتخزين الملفات المؤقتة temporary files داخل الذاكرة الرئيسية للكمبيوتر . ويفقد الملف المؤقت بمجرد الانتهاء من تنفيذ البرنامج الذى أنتجه . وعلى هذا ... فالملفات المؤقتة أقل نفعا ، أو أقل استخداما من الملفات الدائمة . ويشار فى البسكال إلى الملفات المؤقتة بأنها ملفات داخلية internal files .

ويمكن تنظيم كل الملفات ، سواء أكانت دائمة أم مؤقتة بإحدى طريقتين ، إما تتابعيا ، أو عشوائيا ( الطريقة الأخيرة تعرف بأن الملف هو ملف اتصال مباشر direct - access وهو اسم أكثر تعبيراً عن تنظيم الملف ) . فى الملف التتابعى Sequential File تخزن مكونات الملف متتابعة واحدة تلو الأخرى . والوصول إلى إحدى مكونات الملف يكون من الضرورى البدء ببداية الملف ، والبحث خلال الملف حتى الوصول إلى المطلوب . وهذا النوع من أنواع الاتصال بطئ ، خاصة إذا ما كان الملف كبيرا . وإنتاج الملفات التتابعية سهل نسبيا ، وهى الملفات الوحيدة التى يمكن استخدامها مع بعض أنواع أوساط التخزين ، وهى الشريط المغناطيسى .

أما فى ملفات الاتصال العشوائى random - access file ، فيمكن الاتصال بإحدى مكونات الملف مباشرة ، دون البحث خلال محتويات الملف منذ بدايته . ومثل هذه الملفات تيسر اتصالا أسرع كثيرا لإحدى مكونات الملف عن الملفات التتابعية ، وذلك بالرغم من أنه من الصعب جدا إنتاج هذه الملفات وصيانتها .

وبسكال ISO القياسى يستخدم الملفات التتابعية فقط ، وذلك بالرغم من أن بعض مترجمات اللغة تستخدم ملفات الاتصال العشوائى . وسوف نناقش الملفات التتابعية ( القياسية ) فقط فى هذا الكتاب .

#### 2. DEFINING A FILE

#### ٢ - تعريف الملف :

أسهل طريقة لتعريف الملف هى اعتبار تعريف الملف كجزء من توضيح متغير له نوع الملف .

والصفة العامة لمثل هذا التعريف هى :

VAR file name : FILE OF type

حيث file name هو معرف يمثل اسم الملف . وتشير type إلى نوع البيانات الخاصة بمكونات الملف الفردية . ويجب أن تكون جميع هذه المكونات من نفس النوع . ويمكن أن تكون من أى نوع ، سواء أكان بسيطاً أم مركباً ، فيما عدا أنها لا يمكن أن يكون لها نوع ملف . وعلى هذا ... فلا يمكن أن يحتوى الملف ملفاً آخر .

مثال (١-١١)

فيما يلى تعريفاً بسيطاً لملف . وفى هذا التعريف مكونات الملف عبارة عن عناصر بيانات بسيطة من النوع الحرفى .

VAR symbols : FILE OF char;

لاحظ أن اسم الملف هو Symbol .

مثال (٢-١١)

اعتبر الآن تعريف ملف ، كل مكون من مكوناته عبارة عن منظومة ذات بعدين .

TYPE table = ARRAY [1..50,1..20] OF real;  
VAR data : FILE OF table;

فى هذه الحالة اسم الملف هو data .

مثال (٣-١١)

فيما يلى مثالا لتعريف ملف ، كل مكون من مكوناته عبارة عن سجل .

TYPE status = (current,overdue,delinquent);  
account = RECORD  
    custname : PACKED ARRAY [1..80] OF char;  
    custno : 1..9999;  
    custtype : status;  
    custbalance : real  
END;  
VAR customers : FILE OF account;

لاحظ أن كل سجل داخل الملف customers يحتوى على منظومة مضغوطة من النوع الحرفى ( أى متغير سلسلة ) وعلى كمية عددية صحيحة ، ونوع بيانات متعدد ، وكمية عددية حقيقية .

أعتبر الآن طريقة أكثر عمومية لتعريف الملفات ، حيث نعرف فيها نوع الملف أولاً ، ثم توضيح متغيراً أو أكثر ، ليكون من نفس هذا النوع للملف . ( سبق أن استخدمنا هذه الطريقة فى تعريف منظومات وسجلات ) . والصفة العامة من هذا النوع لتعريف الملف ، هى :

TYPE name = FILE OF type

حيث تشير type إلى نوع بيانات كل مكون من مكونات الملف . ويجب أن يكون هذا النوع سبق تعريفه .

### مثال (١١-٤)

فيما يلى مثالا لتعريف ملف ، موجود فيه ملفان مختلفان ، لهما مكونات من السجلات التى لها نفس النوع .

```
TYPE status = (current,overdue,delinquent);
account = RECORD
    custname : PACKED ARRAY [1..80] OF char;
    custno : 1..9999;
    custtype : status;
    custbalance : real
END;
customers = FILE OF account;
VAR oldcustomers,newcustomers : customers;
```

وعلى هذا .. فإن كلا من newcustomers, oldcustomers ملفا من نوع customers . ومكونات كل ملف هى سجلات من نوع account كما هو معرف أعلاه .

لاحظ أن طول الملف لا يحدد على الإطلاق كجزء من تعريف الملف . ويختلف هذا الموقف عن أنواع البيانات المرتبة الأخرى ، حيث تحدد عدد المكونات بوضوح ( فى حالة المنظومات ) أو ضمنا ( فى حالة السجلات ) ، وذلك داخل تعريف نوع البيانات . وكقاعدة عامة ... يتحدد أقصى طول للملف بواسطة السعة الطبيعية للوسط الذى يخزن عليه الملف .

يجب أن تمرر الملفات الخارجية إلى برنامج البسكال كمؤشرات ، ويتحقق ذلك بوضع أسماء الملفات الخارجية فى عنوان البرنامج ، محصورة بين قوسين . وإذا كان هناك أكثر من ملف واحد ، فتفصل الملفات عن بعضها بواسطة فواصل . مواصفات المؤشرات هذه توجد بالإضافة إلى تعريفات الملف المطلوبة والتى يجب أن توجد داخل المجموعة الرئيسية من البرنامج .

### مثال (١١-٥)

افرض أن أحد برامج البسكال يستخدم ملفاً خارجياً اسمه customers ، والموصوف فى مثال (١١-٣) . فيما يلى تخطيطا هيكلياً لهذا البرنامج :

```
PROGRAM sample(customers);
.
.
.
TYPE status = (current,overdue,delinquent);
account = RECORD
    custname : PACKED ARRAY [1..80] OF char;
    custno : 1..9999;
    custtype : status;
    custbalance : real
END;
```

```
VAR customers : FILE OF account;
.
.
BEGIN (* main action statements *)
.
.
(* process the records within customers *)
.
.
END.
```

لاحظ أن اسم الملف customers موجود فى عنوان البرنامج بين قوسين . ويتبع ذلك تعريف الملف ، والذي يظهر داخل جزء التوضيح من المجموعة الأساسية للبرنامج .

ويوجد فى البسكال ملفان ( سبق توضيحهما ) قياسيان ، هما : input ، output ، وعن طريقهما يتم توجيه معلومات من النوع الحرفى من وحدة مدخلات ، وإلى وحدة مخرجات على التوالى . يجب أن يتواجد اسم الملفين هذين فى عنوان البرنامج ، إذا ما كان البرنامج سيقوم بالاتصال بهما . وحيث إنهما من معالم اللغة القياسية ، فهما ليسا على أية حال فى حاجة لأن يعرفا بوضوح داخل البرنامج . وفى واقع الأمر ... إننا استخدمنا هذين الملفين القياسيين خلال هذا الكتاب ، وسوف نقول المزيد عن ملفات المدخلات والمخرجات سابقة التوضيح فى قسم ( ٨ من هذا الفصل ) .

### مثال (١١-٦)

افرض الآن أن برنامج البسكال الموضح تخطيطه الهيكلى فى مثال ( ١١-٥ ) تم تعديله ليستخدم ملفات output ، input ، قياسية . فيما يلى تخطيطا مناسباً للبرنامج .

```
PROGRAM sample(input,output,customers);
.
.
TYPE status = (current,overdue,delinquent);
account = RECORD
    custname : PACKED ARRAY [1..80] OF char;
    custno : 1..9999;
    custtype : status;
    custbalance : real
END;
VAR customers : FILE OF account;
.
.
BEGIN (* main action statements *)
.
.
(* process the records within customers *)
.
.
END.
```

وهذا التخطيط متطابق مع نظيره الموجود فى مثال ( ١١-٥ ) ، باستثناء وجود اسمى ملفى input ، output كمؤشرين فى عنوان البرنامج .

ويجب أن يكون واضحاً أنه ليس هناك حاجة لظهور ملفى input ، output معاً دائماً فى عنوان البرنامج . فيجب تحديد الملفات المطلوب وجودها فعلاً فقط . فإذا ما كان البرنامج ينتج مخرجات ، دون الحاجة إلى بيانات مدخلات ، فيجب أن يحتوى عنوان البرنامج على اسم ملف بيانات output ، ولأحاجة إلى اسم ملف input .

### 3. CREATING A FILE

### ٣ - إنتاج الملف

نركز انتباهنا الآن إلى إنتاج ملف فى البسكال . يجب أن نرى أولا كيفية الاتصال بعنصر بيانات فردى داخل الملف . ويتحقق ذلك عن طريق احتياطى الملف file buffer ، والذي يسمح بقيمة ( أو قيم ) تصاحب مكونات ملف فردى لتنتقل بين الملف وبرنامج المضيف .

وفى واقع الأمر فإن احتياطى الملف file buffer هو نوع خاص من أنواع المتغيرات ، والتي تعرف تلقائيا عندما يتم تعريف الملف المناظر . ويجب أن يكون لاحتياطى الملف نفس اسم الملف ويكون له سهم رأسى مضاف فى نهايته . وعلى هذا ... فإذا ما كان اسم الملف file name يمثل اسم الملف ، فإن احتياطى الملف المناظر يسمى file name . ويجب أن يكون نوع احتياطى الملف هو نفس نوع مكونات الملف .

#### مثال (٧-١١)

افترض أن أحد برامج البسكال يحتوى على تعريف ملف اسمه data . وعلى هذا ... فإن احتياطى الملف المناظر يسمى data ، فإذا ما كانت مكونات الملف سجلات ، فإن data يمثل مثل هذه السجلات . وعند إنتاج ملف ، فإن الفكرة الأساسية هى تحديد عناصر البيانات التى تكون أحد مكونات ملف لاحتياطى الملف ، وبعد ذلك تنقل هذه القيم من احتياطى الملف إلى الملف نفسه . ويتم إنتاج محتويات الملف بتكرار هذه العملية لكل مكون من مكونات الملف .

وأول خطوة فى إنتاج الملف هى إعداد الملف للكتابة . ويتم تحقيق ذلك باستخدام إجراء قياسى اسمه rewrite . وعلى هذا فإننا نبدأ بكتابة مايلى :

```
rewrite(file name);
```

فإذا ما كان إنتاج الملف المحدد يحدث لأول مرة ، فإن عبارة rewrite تحدد ببساطة بداية الملف . أما اذا ما كان الملف موجودا بالفعل بنفس الاسم المعطى ، فإن تأثير عبارة rewrite ذلك هو حذف كل المعلومات الموجودة داخل الملف مثلا ، ثم تحديد بداية للملف الجديد .

وتحدد عناصر البيانات التى يتكون فيها مكونات ملف فردى لاحتياطى الملف بنفس الطريقة التى يتم تحديد عناصر بيانات بها للمتغيرات التى تناظرها . فإذا ما احتوت المكونات على عناصر بيانات من النوع البسيط ، فإن كل تحديد لاحتياطى الملف يحتوى على قيمة فردية . ويجب بالطبع أن تكون هذه القيم متوافقة مع نوع احتياطى الملف . ( عادة ماتكون عناصر البيانات من نفس نوع احتياطى الملف ) .

إذا ما احتوى الملف على مكونات من النوع المرتب ( أى على منظومات أو سجلات ) ؛ فيكون احتياطى الملف متغيرا مناظرا من النوع المرتب ، وعلى هذا ... يمكن تحديد قيمة المتغير المناظر من النوع المرتب للاحتياطى ، أو يمكن تحديد عنصر بيانات فردى منفصلا . وفى الحالة الأخيرة فإن التحديدات الفردية لمكونات الاحتياطى يجب أن تتبع القواعد المعتادة لتحديد عناصر البيانات لأنواع البيانات المرتبة .

بعد تحديد عناصر البيانات اللازمة لاحتياطى الملف ، تنقل هذه المعلومات إلى الملف بواسطة إجراء قياسى اسمه Put . وعلى هذا يمكننا كتابة مايلى :

```
file name↑ := . . . ;
put(file name);
```

فإذا ما تكررت تنفيذ هاتين العبارتين ؛ فينتج عن ذلك كتابة تسلسل من عناصر البيانات في الملف . تخزن عناصر البيانات هذه داخل الملف بنفس الترتيب الذي كتبت به . وفي كلمات أخرى ... فإن عناصر البيانات المصاحبة للمكون الحالي توضع في نهاية الملف ، تالية لعناصر البيانات ( قيم مكونات ) التي سبق كتابتها من قبل .

### مثال (٨-١١)

فيما يلي مثالا لبرنامج بسكال ينتج ملفا اسمه data . يحتوى هذا الملف على مكونات من النوع البسيط ( أعداد حقيقية موجبة ) .

```
PROGRAM createfile1(input,output,data);
VAR data : FILE OF real;
    item : real;
    count,total : 1..maxint;
BEGIN
    write('How many values will be entered? ');
    readln(total);
    rewrite(data);
    FOR count := 1 TO total DO
        BEGIN
            readln(item);
            data+ := item;
            put(data)
        END
    END.
```

لاحظ أن هذا البرنامج يتسبب في إدخال بيانات من وحدة مدخلات ( أى لوحة مفاتيح ) ، ثم يكتب هذه البيانات على ملف جديد اسمه data .

ويبدأ البرنامج بسؤال المستفيد عن عدد القيم ( أى عدد مكونات الملف ) التي يراد إدخالها . وبعد قراءة هذه القيمة داخل الكمبيوتر ، يتم إعداد الملف الجديد عن طريق عبارة rewrite ، وبعد ذلك يستخدم البرنامج بنية FOR - TO لقراءة قيمة كل مكون من مكونات الملف من وحدة المدخلات ، ثم كتابتها في الملف .

لاحظ أن هذا البرنامج يتطلب أن يعرف المستفيد مسبقا عدد عناصر البيانات ( أى عدد مكونات الملف ) التي سوف تضاف الى الملف . وهذا غير عملي بعض الشيء ، لكنه يكون عمليا إذا ما كان الملف طويلا نسبيا . والمثال التالي يقدم طريقة أفضل .

### مثال (٩-١١)

فيما يلي صيغة أخرى للبرنامج الموجود في مثال (٨ - ١١) . يمثل هذا البرنامج تعديلا للصيغة السابقة ، بمعنى أن عدد عناصر البيانات ليس هناك حاجة لمعرفته مسبقا . ويستمر المستفيد ببساطة في إدخال البيانات حتى تظهر قيمة سالبة تحدد إنتهاء الملف ( لاحظ أن هذا المخطط يوجد عليه قيد ، وهو أن مكونات الملف الفعلية يجب ألا تكون سالبة ) .

```
PROGRAM createfile2(input,data);
VAR data : FILE OF real;
    item : real;
BEGIN
```

(تكملة البرنامج في الصفحة التالية)



```

rewrite(data);
readln(item);
WHILE item >= 0 DO
  BEGIN
    data↑ := item;
    put(data);
    readln(item)
  END
END.

```

لاحظ أن عنوان البرنامج يحتوى على مؤشرات ملفات input , output . ولا يحتاج هذا البرنامج الى استخدام ملف output قياسى ، وذلك لعدم وجود ملفتنا يكتبها البرنامج . ( قارن ذلك بالبرنامج الموجود فى مثال ١١-٨ ، والذي ينتج ملفتنا لإجمالي عدد عناصر البيانات ) .

إذا ماكانت مكونات الملف مرتبة ، فقد يكون من المرغوب فيه إدخال عنصر بيانات فردى لكل مكون من مكونات الملف بواسطة إجراء . وفى مثل هذه الحالات يجب أن يمر متغير مرتب من نفس نوع احتياطى الملف إلى الإجراء كمؤشر فعلى . ويكون المؤشر الرسمى المناظر مؤشراً متغيراً من نفس نوع احتياطى الملف أيضاً . وبعد الاتصال بالإجراء يجب أن تحدد البيانات التى يمثلها المؤشر الرسمى لاحتياطى الملف ، مع السماح بكتابة مكونات الملف فى الملف . والتفاصيل موضحة فى المثال التالى :

#### مثال (١١-١٠)

إنتاج ملف لنظام فواتير العملاء . اعتبر الآن برنامجاً ينتج ملفاً مكوناً من سجلات فواتير عملاء كما هو معرف فى مثال (١١-٥) . يظهر توضيح البرنامج على النحو التالى :

```

TYPE status = (current,overdue,delinquent);
account = RECORD
  custname : PACKED ARRAY [1..80] OF char;
  custno : 1..9999;
  custtype : status;
  custbalance : real
END;
VAR customeracct : account;
customers : FILE OF account;
count : 0..80;

```

هذه التوضيحات هى نفسها مثل التوضيحات فى مثال (١١-٥) ، فيما عدا توضيحات متغير السجل الإضافى customeracct ، والذي يستخدم كمؤشر ، والمتغير صحيح النوع count .

افرض أننا نستخدم إجراء فى إدخال عناصر البيانات الفردية لكل مكون من مكونات الملف ( فى هذه الحالة لكل سجل ) . سوف يقرأ الإجراء قيم اسم العميل ، ورقم العميل ، وموازنة العميل فى الكمبيوتر . ويمكن عند ذلك تحديد نوع العميل ( حالته status ) طبقاً لقيمة موازنة العميل ( أو لاي معيار آخر ) .

ويمكن كتابة إجراء المدخلات input على النحو التالي :

```
PROCEDURE readinput(VAR info: account);
(* Enter input data for one record *)
BEGIN
  WITH info DO
    BEGIN
      write('Customer number: ');
      readln(custno);
      IF custno <> 9999 THEN
        BEGIN
          write('Name: ');
          count := 0;
          WHILE NOT eoln DO
            BEGIN
              count := count + 1;
              read(custname[count])
            END;
          writeln;
          write('Current balance: ');
          readln(custbalance);
          writeln;
          custtype := current
        END
      END (* WITH info *)
    END; (* readinput *)
```

لاحظ أن info هو مؤشر رسمي من نفس نوع account . وعلى هذا ... فإن info متغير من نوع السجل . لاحظ أيضا أن الإجراء يقرأ اسم العميل أولا ، ثم يستمر إذا ما كان رقم العميل لا يساوي 9999 فقط . وعلى هذا ... فإن رقم العميل 9999 يقدم طريقة مقننة لإيقاف العمل .

والجزء الرئيسي للبرنامج بسيط ، حيث يتم إعداد الملف الجديد customers ، ثم قراءة أول سجل ، وبعد ذلك يتم تكرار الخطوات التالية ، طالما أن رقم العميل لا يساوي 9999 .

١ - نحدد أحدث البيانات ( أحدث سجل ) لاحتياطي الملف .

٢ - نكتب احتياطي الملف ( أحدث سجل ) في الملف .

٣ - نقرأ السجل الجديد .

وفيما يلي برنامجا كاملا لأداء ذلك .

```
PROGRAM createfile3(input,output,customers);
(* THIS PROGRAM CREATES A FILE OF CUSTOMER RECORDS *)
TYPE status = (current,overdue,delinquent);
account = RECORD
  custname : PACKED ARRAY [1..80] OF char;
  custno : 1..9999;
  custtype : status;
  custbalance : real
END;
```

(تكملة البرنامج في الصفحة التالية)

```

VAR customeracct : account;
    customers : FILE OF account;
    count : 0..80;

PROCEDURE readinput(VAR info : account);
(* Enter input data for one record *)
BEGIN
    WITH info DO
        BEGIN
            write('Customer number: ');
            readln(custno);
            IF custno <> 9999 THEN
                BEGIN
                    write('Name: ');
                    count := 0;
                    WHILE NOT eoln DO
                        BEGIN
                            count := count + 1;
                            read(custname[count])
                        END;
                    writeln;
                    write('Current balance: ');
                    readln(custbalance);
                    writeln;
                    custtype := current
                END
            END (* WITH info *)
        END; (* readinput *)

    BEGIN (* main action statements *)
        rewrite(customers);
        readinput(customeracct);
        WHILE customeracct.custno < 9999 DO
            BEGIN
                customers↑ := customeracct;
                put(customers);
                readinput(customeracct)
            END
        END.
    
```

#### ٤ - المزيد عن عبارة WRITE : WRITE STATEMENT

من الأمثلة السابقة رأينا أن العبارات :

```

file name↑ := identifier;
put(file name);

```

عادة ماتستخدم فى كتابة مكونات ملف فى الملف . ويمكن تحقيق نفس الشئ بسهولة أكثر ، وذلك بكتابة :

```
write(file name , identifier);
```

حيث يمثل identifier متغيرا أو أى معرف تحدد قيمته ( أو قيمة فى حالة احتياطى ملف من النوع المرتب ) لاحتياطى الملف . ولايسمح هذه البديل لنا باستبدال عبارتين بعبارة واحد فقط ، بل إنه يلغى أيضا الحاجة لأى إشارة صريحة لاحتياطى الملف .

### مثال (١١-١١)

اعتبر مرة أخرى برنامج البسكال البسيط الموضح في مثال (١١-٩) . يمكن كتابة هذا البرنامج على النحو التالي أيضا :

```
PROGRAM createfile4(input,data);
VAR data : FILE OF real;
    item : real;
BEGIN
    rewrite(data);
    readln(item);
    WHILE item >= 0 DO
        BEGIN
            write(data,item);
            readln(item)
        END
    END.
END.
```

لاحظ أننا استبدلنا العبارتين :

```
data := item;
put(data);
```

في المثال السابق بعبارة write واحدة ، وهي :

```
write(data,item);
```

### مثال (١٢-١١)

دعنا نعود الآن إلى برنامج نظام الفواتير الموجود في مثال (١١-١٠) . يمكن تعديل هذا البرنامج ليأخذ الشكل التالي :

```
PROGRAM createfile5(input,output,customers);

(* THIS PROGRAM CREATES A FILE OF CUSTOMER ACCOUNTS *)

TYPE status = (current,overdue,delinquent);
    account = RECORD
        custname : PACKED ARRAY [1..80] OF char;
        custno : 1..9999;
        custtype : status;
        custbalance : real
    END;
VAR customeracct : account;
    customers : FILE OF account;
    count : 0..80;

PROCEDURE readinput(VAR info : account);
(* Enter input data for one record *)
BEGIN
    WITH info DO
        BEGIN
```

(تكملة البرنامج في الصفحة التالية)

```

write('Customer number: ');
readln(custno);
IF custno <> 9999 THEN
  BEGIN
    write('Name: ');
    count := 0;
    WHILE NOT eoln DO
      BEGIN
        count := count + 1;
        read(custname[count])
      END;
    writeln;
    write('Current balance: ');
    readln(custbalance);
    writeln;
    custtype := current
  END
END (* WITH info *)
END; (* readinput *)

BEGIN (* main action statements *)
  rewrite(customers);
  readinput(customeracct);
  WHILE customeracct.custno < 9999 DO
    BEGIN
      write(customers,customeracct);
      readinput(customeracct)
    END
  END
END.

```

استبدلنا فى هذه الحالة العبارتين :

```

customers := customeracct;
put(customers);

```

الموجودتين فى المجموعة الرئيسية للبرنامج بعبارة write واحدة ، وهى :

```

write(customers,customeracct);

```

ويمكن أن تحتوى عبارة write على معرفات متعددة . إذا كانت هناك رغبة فى ذلك فى هذه الحالة تظهر عبارة write فى الصورة التالية :

```

write(file name, identifier 1, identifier 2, . . . , identifier n);

```

والتي تكافئ :

```

BEGIN
  write(file name, identifier 1);
  write(file name, identifier 2);
  .
  .
  .
  write(file name, identifier n)
END;

```

ويجب أن يكون مفهوماً على أية حال أن نوع البيانات الذى يصاحب كل معرف يجب أن يكون من نفس نوع احتياطي الملف ، أو متوافقاً معه .

أكثر من ذلك ... فالعناصر المسروقة فى عبارة write ليس هناك حاجة لأنها تكون معرفات فردية . ويمكن أن تكون تعبيرات أيضاً ، على أن تكون أنواع البيانات الناتجة منها متوافقة مع نفس نوع احتياطي الملف .

وعلى هذا ... فإن الصيغة العامة لعبارة write يمكن كتابتها على النحو التالى :

```
write(file name, output item 1, output item 2, . . . , output item n);
```

### مثال (١١-١٣)

فيما يلي برنامج بسكال بسيطاً ، ينتج ملفاً يحتوى على أول ١٠٠ عدد صحيح ومربعاتها وجذورها التربيعية .

```
PROGRAM createfile6(data);

(* GENERATE A FILE CONTAINING THE FIRST 100 INTEGERS,
   THEIR SQUARES AND THEIR SQUARE ROOTS *)

VAR data : FILE OF real;
    value : real;
    count : 1..100;

BEGIN (* main action statements *)
    rewrite(data);
    FOR count := 1 TO 100 DO
        BEGIN
            value := count;
            write(data,value,sqr(value),sqrt(value))
        END
    END;
```

لاحظ أن عناصر البيانات مخزنة على هيئة كميات حقيقية . لاحظ أيضاً أن كل عدد صحيح يتبعه مربعه وجذره التربيعي . وأخيراً لاحظ أن البرنامج لا يستخدم أى من الملفين القياسيين input أو output .

## 5. READING A FILE

### ٥ - قراءة الملف :

عملية قراءة ملف هي بالضرورة صورة في المرأة لعملية إنتاج الملف . أول خطوة هي إعداد الملف للقراءة باستخدام إجراء قياسى اسمه reset . ويتم تحقيق ذلك بكتابة مايلي :

```
reset(file name);
```

إذا كان الملف يحتوى على مكون واحد . أو أكثر من مكون واحد ، فتتسبب عبارة reset فى قراءة قيمة أول مكون من مكونات الملف ، وتحديد ما احتياطي الملف . بالإضافة إلى ذلك ... تكون دالة بوليان القياسية eof ( نهاية الملف ) خاطئة false . فإذا كان الملف فارغاً - على أية حال - فإن احتياطي الملف يظهر غير معرف ، وتكون قيمة eof حقيقية true .

وبعد قراءة إحدى مكونات الملف وتحديد ما لاحتياطي الملف ، يمكن تشغيل محتويات احتياطي الملف إذا كانت هناك رغبة في ذلك . فمثلا يمكن تحديد مكونات احتياطي الملف للمتغير ، أو طباعة قيمتها ، أو كتابتها في ملف .. الخ . بعد ذلك يمكن الحصول على المكون الثاني للملف باستخدام إجراء قياسى اسمه `get` . وعلى هذا ... يحتوى البرنامج بصورة تقليدية على تتابع التعليمات التالية :

```
(* statement to process the contents of file name↑ *)
.
.
.
get(file name);
```

( لاحظ أن أولى التعليمات ممثلة بواسطة تعليق ، حيث إنها ليست معرفة بوضوح ، أى أن طبيعتها الدقيقة تتغير من برنامج لبرنامج آخر ) . وتتكرر هذه التعليمات بصورة تقليدية حتى يتم قراءة وتشغيل كل مكونات الملف ، وعند ذلك تصبح قيمة الدالة `eof` صحيحة `true` . وطريقة تحقيق ذلك موضحة في المثال التالى :

### مثال (١١-١٤)

فيما يلى برنامج يسكال بسيطاً يقرأ ملفاً يحتوى على أعداد حقيقية . ويكتب كل عدد فى وحدة مخرجات قياسية كما يقرأ من الملف .

( لاحظ أنه يمكن استخدام هذا البرنامج فى تشغيل الملفات التى تم إنتاجها فى مثالى (٨ - ١١) و (٩ - ١١) ) .

```
PROGRAM readfile1(output,data);

(* THIS PROGRAM READS REAL NUMBERS FROM A DATA FILE
AND WRITES THEM TO A STANDARD OUTPUT DEVICE *)

VAR data : FILE OF real;
    item : real;

BEGIN
    reset(data);
    WHILE NOT eof(data) DO
        BEGIN
            item := data↑;
            write(item);
            get(data)
        END
    END.
END.
```

هناك عدة ملاحظات فى هذا المثال . لاحظ أولاً أن عنوان البرنامج يحتوى على مؤشرات ملف `data` , `output` ، ولا يحتوى على مؤشر `input` ( حيث إنه لا يوجد نقل معلومات من وحدة مدخلات قياسية ) . لاحظ ثانياً طريقة استخدام المعارف القياسية `reset` , `eof` , `get` . يحتوى كل معرف على بيانات اسم ملف محصورة بين قوسين . ولاحظ أخيراً أن كل مكون من مكون الملف يتم تشغيله بتحديد محتويات احتياطي الملف أولاً للمتغير من النوع الحقيقى ، ثم بعد ذلك كتابة قيمة المتغير فى وحدة المخرجات القياسية .

ويجب أن يكون مفهوماً أنه ليس هناك حاجة لمعرفة إجمالى عدد مكونات الملف معرفة مسبقاً . وسوف تقرأ المكونات على التوالى بدءاً ببداية الملف ، وتستمر حتى انتهاء الملف . وهذه الطريقة شائعة جداً فى تشغيل محتويات الملف التتابعى .

## 6. MORE ABOUT THE READ STATEMENT : المزيد عن عبارة READ : ٦ -

عند القراءة من ملف تتابعى ، فمن الشائع تحديد محتويات احتياطى الملف لمتغير من نوع مناسب ، ثم الحصول على المكون التالى للملف ( أى تحديد محتويات المكون التالى للملف لاحتياطى الملف ) .

يمكن كتابة هاتين العمليتين على النحو التالى :

```
variable name := file name↑ ;
get(file name);
```

وحيث إنه يتكرر استخدام هاتين العبارتين ، فإن لغة البسكال تقدم طريقة أسهل لتحقيق نفس الشئ ، وذلك باستخدام عبارة read . وعلى هذا ... فيمكننا كتابة مايلى :

```
read(file name, variable name);
```

بدلا من العبارتين السابقتين ، لاحظ أن احتياطى الملف لا يظهر فى عبارة read .

مثال (١١-١٥)

فيما يلى صيغة أخرى للبرنامج الموجود فى مثال (١١-٤) . هذه الصيغة تستخدم عبارة read ، بدلا من استخدام get يسبقها عبارة تحديد .

```
PROGRAM readfile2(output,data);
(* THIS PROGRAM READS REAL NUMBERS FROM A DATA FILE
   AND WRITES THEM TO A STANDARD OUTPUT DEVICE *)
VAR data : FILE OF real;
    item : real;
BEGIN
    reset(data);
    WHILE NOT eof(data) DO
        BEGIN
            read(data,item);
            writeln(item)
        END
    END.
END.
```

داخل دورة DO - WHILE تؤدي عبارة read مايلى :

١ - تحديد القيمة الحالية لاحتياطى الملف للمتغير الحقيقى item .

٢ - قراءة المكون التالى للملف وتحديد احتياطى الملف .

وعند ذلك تتسبب عبارة write فى كتابة القيمة الحالية للمتغير item فى وحدة مخرجات قياسية ( أى عن طريق طابع الأسطر ، أو نهاية طرفية للمشاركة الزمنية ) .



يمكن أن توجد متغيرات متعددة في عبارة read إذا كانت هناك رغبة في ذلك . وعلى هذا ... فيمكن كتابة عبارة read على النحو التالي :

```
read(file name, variable name 1, variable name 2, . . . , variable name n);
```

والذى يكافئ مايلى :

```
BEGIN
    read(file name, variable name 1);
    read(file name, variable name 2);
    .
    .
    .
    read(file name, variable name n)
END;
```

مرة أخرى ، يجب أن يكون مفهوما أن نوع البيانات الذى يصاحب كل متغير يجب أن يكون من نفس نوع احتياطي الملف ، أو من نوع متوافق معه .

مثال (١١-١٦)

رأينا في مثال (١١ - ١٣) برنامجا بلغة البسكال ، ينتج ملفا محتويا على تسلسل من الأعداد ومربعاتها وجنورها التربيعية . دعنا نعتبر الآن برنامجا يقرأ هذا الملف . مثل هذا البرنامج مدون أدناه :

```
PROGRAM readfile3(output,data);
(* THIS PROGRAM READS A DATA FILE CONTAINING THE FIRST
100 INTEGERS, THEIR SQUARES AND THEIR SQUARE ROOTS *)
VAR data : FILE OF real;
    value,square,root : real;
BEGIN . (* main action statements *)
    reset(data);
    WHILE NOT eof(data) DO
        BEGIN
            read(data,value,square,root);
            writeln(value:6:2,square:8:2,root:6:2)
        END
    END.
END.
```

لاحظ أن هذا البرنامج أكثر عمومية عن مثيله الموجود في مثال (١١ - ١٣) ، حيث إنه ليس مقيدا على ملف سبق تحديد حجمه . لاحظ أيضا مظهر المتغيرات المتعددة في عبارة read .

## 7. UPDATING A FILE

## ٧ - تجديد الملف

يمكن إنتاج ملف تتابعي داخل برنامج البسكال كملف مدخلات فقط أو ملف مخرجات فقط ، ولا يمكن إنتاج ملفي مدخلات ومخرجات في نفس البرنامج . ( لاحظ على أية حال أنه يمكن معاملة ملف معين كملف مدخلات في

برنامج ، وكلف مخرجات فى برنامج آخر ) . وعلى هذا ... فليس من الممكن قراءة مكونات ملف وتعديلها وكتابة المكونات المعدلة على نفس الملف . وبالمثل ليس من الممكن قراءة ملف حتى نهايته ، ثم إضافة سجلات إضافية على نفس الملف .

تميل هذه القيود إلى تعقيد عملية تجديد الملف . ويمكن تجديد الملف بطريقة مباشرة نسبية على أية حال ، وذلك باستخدام ملفين داخل البرنامج . أحد الملفين يكون الملف الأصلى المراد تجديده ( الملف القديم ) . وسوف يعامل هذا الملف كملف مدخلات . أما الملف الثانى ( الملف الجديد ) ، فإنه يحتوى على المعلومات الجديدة ، وبعض هذه المعلومات ينسخ مباشرة من الملف القديم ، والبعض الآخر يتم إدخاله من وحدة مدخلات قياسية ( مثل نهاية طرفية للمشاركة الزمنية ) . وسوف يعامل على ذلك الملف الجديد على أنه ملف مخرجات .

ولتجديد ملف قديم ، فإننا نجرى الخطوات التالية لكل مكون من مكونات الملف .

١ - يقرأ مكون الملف داخل ذاكرة الكمبيوتر ، ويحدد ما إذا كان سيجدد أم لا . ويمكن أن يتحقق ذلك بمقارنة أحد الأدلة لعنصر بيانات ، مثل رقم العميل مع قيمة تم إدخالها من وحدة مدخلات قياسية . ( تحدد القيمة التى يتم إدخالها من وحدة المدخلات القياسية المكون الثانى الذى ستجرى عليه عملية التجديد ) .

أ - إذا كان مكون الملف الحالى لا يحتاج إلى تجديد ، فيتم نسخه مباشرة فى الملف الجديد .

ب - إذا كان التجديد مطلوباً ، فيتم إدخال المعلومات الجديدة من وحدة المدخلات ودمجها مع المعلومات القديمة بالطريقة المناسبة ، ثم يكتب مكون الملف الجديد فى الملف الجديد . وبعد ذلك يتم إدخال قيمة أخرى من وحدة المدخلات تحدد مكون الملف الثانى المطلوب تجديده .

٢ - تتكرر هذه العملية حتى تقرأ كل مكونات الملف من الملف القديم ( أى حتى يتم الوصول إلى نهاية الملف ) .

٣ - إذا كان مطلوباً إضافة مكونات جديدة إلى الملف الأصلى ، فيجب إدخالها عند ذلك الوقت من وحدة المدخلات بالترتيب المناسب . ويكتب كل مكون من مكونات الملف الجديدة فى الملف الجديد كما يدخل من وحدة المدخلات تماماً .

ولكى تعمل هذه الخطوات بطريقة مناسبة يجب إدخال المعلومات الجديدة متتالية من وحدة المدخلات ، وفى الترتيب الصحيح . ويجب أن يتوافق الترتيب مع الترتيب المستخدم فى تخزين مكونات الملف داخل الملف الأصلى . فمثلاً إذا كانت مكونات الملف هى سجلات العملاء المخزنة فى ترتيب تصاعدي طبقاً لرقم العميل ، فيجب أن تدخل المعلومات الجديدة ترتيباً تصاعدياً طبقاً لرقم العميل أيضاً .

مثال (١١-١٧)

نسخ ملف . فيما يلى برنامجاً موجزاً يقتضب فى نسخ مكونات ملف اسمه old file فى ملف آخر اسمه new

file

```
PROGRAM sample(newfile,oldfile);
(* THIS PROGRAM COPIES THE CONTENTS OF ONE FILE TO ANOTHER *)
TYPE account = RECORD
    custno : 1..9999;
    oldbalance : real;
    newbalance : real;
    payment : real;
END;
```

(تكملة البرنامج فى الصفحة التالية)

```

VAR newfile,oldfile : FILE OF account;
    newaccount,oldaccount : account;
BEGIN (* main action block *)
    reset(oldfile);
    rewrite(newfile);
    WHILE NOT eof(oldfile) DO
        BEGIN
            read(oldfile,oldaccount);
            newaccount := oldaccount;
            write(newfile,newaccount)
        END
    END.

```

وليس من الضرورى فى واقع الامر وجود متغيرين منفصلين من نوع السجل newaccount , oldaccount داخل تخطيط البرنامج . فأحد هذين المتغيرين يكون كافيا . وعلى هذا ... فإن المجموعة الإجرائية الرئيسية يمكن أن تكتب على النحو التالى :

```

BEGIN (* main action block *)
    reset(oldfile);
    rewrite(newfile);
    WHILE NOT eof(oldfile) DO
        BEGIN
            read(oldfile,newaccount);
            write(newfile,newaccount)
        END
    END
END

```

وعلى أية حال ... فإن استخدام متغيرين مختلفين يميز الطريقة التى تنقل بها المكونات الفردية للملف من ملف إلى آخر ، حيث إن محتويات oldaccount يمكن تعديلها بطريقة معينة قبل نقلها إلى newaccount .

مثال (١١-١٨)

دعنا نعتبر الآن عملية نقل إحدى المكونات ، أو العديد من المكونات ، إلى ملف موجود فعلا . ويكون ضروريا مرة أخرى قراءة كل المكونات من oldfile فى newfile . وبعد الانتهاء من نقل كل المكونات القديمة ، يتم إدخال المكونات الجديدة من لوحة المفاتيح . وتستمر هذه العملية حتى يتم إدخال القيمة 9999 كقيمة للمتغير custno .

وفيما يلى تخطيط البرنامج :

```

PROGRAM sample(input,output,newfile,oldfile);
(* TRANSFER THE CONTENTS OF ONE FILE TO ANOTHER,
   THEN APPEND ADDITIONAL COMPONENTS TO THE NEW FILE *)
TYPE account = RECORD
    custno : 1..9999;
    oldbalance : real;
    newbalance : real;
    payment : real;
END;

```

(تكملة البرنامج فى الصفحة التالية)

```

VAR newfile,oldfile : FILE OF account;
    newaccount : account;
    custno : 1..9999;

PROCEDURE readinput(VAR newaccount : account);
(* enter input data for one record *)
BEGIN
    .
    .
    (* enter input data from the keyboard for each record *)
    .
    .
END;

BEGIN (* main action block *)
    reset(oldfile);
    rewrite(newfile);

    WHILE NOT eof(oldfile) DO (* transfer old records *)
        BEGIN
            read(oldfile,newaccount);
            write(newfile,newaccount)
        END; (* transfer of old records *)

        write('Customer number: ');
        readln(custno);
        WHILE custno < 9999 DO (* enter new records *)
            BEGIN
                readinput(newaccount);
                write(newfile,newaccount);
                write('Customer number: ');
                readln(custno)
            END (* enter new records *)
        END.
    END.

```

لاحظ أن كل مكون جديد من مكونات الملف ( كل سجل جديد ) يمرر من الإجراء readinput إلى الجزء الرئيسى للبرنامج كمؤشر من نوع المتغير .

### مثال (١١-١٩)

فيما يلى هيكل البرنامج بسكال يقرأ مكونات ملف متتالية من ملف اسمه oldfile ، وتجديدها إذا كان هذا مطلوباً ، ثم نسخها فى الملف المسمى newfile . ويستمر تجديد المكونات المختلفة حتى تدخل قيمة 9999 كقيمة للمتغير custno . وأي مكونات تظل فى الملف القديم oldfile تنقل عند ذلك مباشرة إلى newfile دون أى فرصة لتجديدها .

لاحظ أن كل مكونات oldfile تنقل إلى newfile . بعضها ينقل بعد تجديده ، وبعضها ينقل كما هو دون أى تغيير .

ولكى تعمل هذه الطريقة بطريقة مناسبة ، فمن الضرورى إدخال القيم المتتالية للمتغير custno فى ترتيب تصاعدى . وهذا يؤكد أن تسلسل التغيير متطابق مع الترتيب المناظر لتخزين مكونات الملف فى oldfile .

```

PROGRAM sample(input,output,newfile,oldfile);

(* THIS PROGRAM READS SUCCESSIVE FILE COMPONENTS FROM A
   DATA FILE, UPDATES EACH COMPONENT IF NECESSARY,
   AND THEN WRITES THE COMPONENTS TO A NEW DATA FILE *)

TYPE account = RECORD
    custno : 1..9999;
    oldbalance : real;
    newbalance : real;
    payment : real;
END;
VAR newfile,oldfile : FILE OF account;
    newaccount,oldaccount : account;
    custno : 1..9999;

BEGIN (* main action statements *)
    reset(oldfile);
    rewrite(newfile);
    write('Customer number: ');
    readln(custno);
    WHILE NOT eof(oldfile) DO
        BEGIN
            read(oldfile,oldaccount);
            newaccount := oldaccount;
            IF (custno < 9999) AND (oldaccount.custno = custno) THEN
                BEGIN
                    .
                    .
                    (* accept appropriate input from the keyboard and
                       make changes in a file component as required *)
                    .
                    .
                    write('Customer number: ');
                    readln(custno)
                END; (* IF custno *)
                write(newfile,newaccount)
            END (* WHILE NOT eof *)
        END
    END.

```

فى بعض الأحيان يكون من الضرورى نقل محتويات الملف إلى إجراء كمؤشر . فى مثل هذه الحالات يجب أن يمرر الملف كمؤشر متغير variable دائما . وهذا يلغى الحاجة لإنتاج نسخة مطولة من الملف ، كما هو الحال إذا ماتم تمرير الملف كمؤشر قيمة . سوف نرى توضيحا لهذه النقطة فى المثال التالى :

### مثال (١١-٢٠)

تجديد نظام فواتير عملاء . نعتبر الآن برنامجا كاملا بلغة البسكال ، يسمح لنا بتجديد ملف بيانات يستخدم فى حفظ نظام فواتير العملاء . يفترض أن مثل هذا التجديد يجرى كل فترة زمنية معينة ( كل شهر مثلا ) .

دعنا نفحص كل السجلات أثناء كل عملية تجديد . يسمح هذا الإجراء بالمدفوعات الحديثة أن تخصم من الحساب الخاص بالعملاء المدنين . ويمكن عند ذلك تغيير حالة الحساب طبقا لهذه المدفوعات .

يحتوى التجديد على تسجيل أحدث المدفوعات وتاريخها ، وحساب الموازنة الجديدة ، وتحديد حالة حساب كل عميل .

افترض أن سجل كل عميل يحتوى على عناصر المعلومات التالية :

الاسم ، ورقم الحساب ( رقم العميل ) ، وحالة الحساب ( جارى أو متأخر أو مقصر ) ، والموازنة السابقة ، والموازنة الجديدة ، ومدفع حاليا ، وتاريخ الدفع . وتوضيحات السجلات المناسبة هي كمايلي :

```
TYPE status = (current,overdue,delinquent);
line = PACKED ARRAY [1..30] OF char;
date = RECORD
    month : 1..12;
    day : 1..31;
    year : 1900..2100
END;
account = RECORD
    name : line;
    custno : 1..9999;
    custtype : status;
    oldbalance : real;
    newbalance : real;
    payment : real;
    paydate : date
END;
```

لاحظ أن توضيحات السجل هي مثل توضيحات المعطاه فى مثال ( ١٠ - ٢١ ) ، فيما عدا ان عنوان العميل حذف هنا للتبسيط .

وتعتبر حالة كل حساب من حسابات العملاء جارية ، إلا إذا لم تكن هناك موازنة . وفى هذه الحالة تعتمد حالة الحساب على حجم أحدث مبلغ تم دفعه ، وتطبيق القواعد التالية فى ذلك .

١ - إذا كان المبلغ المدفوع حالياً أكبر من صفر ، وأقل من ١٠ ٪ من الموازنة السابقة ، فيعتبر الحساب متأخراً .

٢ - إذا كانت هناك موازنة قائمة ، وكانت القيمة المدفوعة حالياً تساوى صفراً ، فيعتبر الحساب مقصراً

( وهذا هو نفس الشئ أيضاً مثل المثال رقم ( ١٠ - ٢١ ) ) .

والعملية الكاملة لكل تجديد للملف تصبح على النحو التالى :

١ - تكتب كل السجلات الموجودة فى ملف البيانات القديم ( وذلك لتقديم نسخة من الملف الأصيل ) .

٢ - يستمر العمل متتالياً خلال الملف القديم ، مع تجديد كل السجلات التى بها موازنات قائمة .

٣ - يكتب كل سجل فى ملف البيانات الجديد . ( لاحظ أن هذا يشمل كل السجلات ، بغض النظر عما إذا حدث لها تجديد أم لا ) . وسوف تكتب السجلات التى أجرى لها تجديد فى صورتها الجديدة المعدلة ، أما السجلات الأخرى ، فسوف تنسخ بنفس الصورة التى كانت عليها فى الملف القديم

٤ - تكتب كل السجلات فى ملف البيانات الجديد .

ويبدو التكوين الهيكلى الشامل لهذا البرنامج على النحو التالى :

```
PROGRAM billing2(input,output,newfile,oldfile);
TYPE status = (current,overdue,delinquent);
line = PACKED ARRAY [1..30] OF char;
date = RECORD
    month : 1..12;
    day : 1..31;
    year : 1900..2100
END;
account = RECORD
    name : line;
    custno : 1..9999;
    custtype : status;
    oldbalance : real;
    newbalance : real;
    payment : real;
    paydate : date
END;
datafile = FILE OF account;
VAR newfile,oldfile : datafile;
newaccount,oldaccount : account;

PROCEDURE writefile(VAR data : datafile);
VAR custaccount : account;
BEGIN
    .
    .
    (* write the contents of the file
       represented by the variable parameter data *)
    .
    .
END;

PROCEDURE update(VAR newaccount : account);
BEGIN
    .
    .
    (* accept appropriate input from the keyboard and
       make changes in a file component as required *)
    .
    .
END;

BEGIN (* main action block *)
    writeln('CUSTOMER BILLING SYSTEM - FILE UPDATE');
    writeln;
    writeln('OLD DATA FILE');
    writeln;
    writefile(oldfile);

    BEGIN (* update the old file *)
```

(تكملة البرنامج فى الصفحة التالية)

```

writeln;
writeln('DATA UPDATE');
writeln;
reset(oldfile);
rewrite(newfile);
WHILE NOT eof(oldfile) DO
  BEGIN
    read(oldfile,oldaccount);
    newaccount := oldaccount;
    IF oldaccount.newbalance > 0 THEN update(newaccount);
    write(newfile,newaccount)
  END (* WHILE NOT eof(oldfile) *)
END; (* file update *)

writeln;
writeln('NEW DATA FILE');
writeln;
writefile(newfile)
END.

```

يستخدم هذا البرنامج إجرائين ، هما : update , writefile . يقرأ أول هذين الإجرائين write file السجلات الموجودة في ملف بيانات ، ثم يكتبها في وحدة مخرجات ( مثل الطابع ) . حيث إن هذا الإجراء يستخدم مع كل من ملفي البيانات ، فأننا نستخدم مؤشراً متغيراً يسمى data داخل توضيح الإجراء . لاحظ أن data يمثل ملفاً من نفس نوع الملف data file كما هو معرف في المجموعة الرئيسية للبرنامج . ويظهر الإجراء الفعلي writefile على النحو التالي :

```

PROCEDURE writefile(VAR data : datafile);
(* THIS PROCEDURE CAUSES THE CONTENTS OF A DATA FILE TO BE WRITTEN OUT *)

VAR custaccount : account;
BEGIN
  reset(data);
  WHILE NOT eof(data) DO
    BEGIN
      read(data,custaccount);
      WITH custaccount DO
        BEGIN
          writeln;
          write('Name: ',name);
          writeln(' Customer Number: ',custno:4);
          writeln;
          write('Old balance: ',oldbalance:7:2);
          write(' Current payment: ',payment:7:2);
          writeln(' New balance: ',newbalance:7:2);
          write('Payment date: ',paydate.month:2,'/');
          writeln(paydate.day:2,'/',paydate.year:4);
          writeln;
          write('Account status: ');
          CASE custtype OF
            current : writeln('CURRENT');
            overdue : writeln('OVERDUE');
            delinquent : writeln('DELINQUENT')
          END;
          writeln
        END (* WITH custaccount *)
      END (* WHILE NOT eof(data) *)
    END;
  END;

```



المنطق مباشر ، وليس في حاجة إلى مناقشة .

اعتبر الآن الإجراء الثاني update . الغرض من هذا الإجراء هو إدخال أحدث مبلغ مدفوع لعميل معين ، وتضبيط موازنة العميل ، وبعد ذلك تحديد الحالة المناسبة لحساب العميل طبقا للموازنة السابقة ، وحجم المبلغ المدفوع حاليا .

ويمكن كتابة إجراء update على النحو التالي :

```
PROCEDURE update(VAR newaccount : account);

(* THIS PROCEDURE ENTERS NEW DATA FROM THE KEYBOARD
   AND THEN UPDATES EACH RECORD AS NECESSARY *)

VAR slash : char;
BEGIN
  WITH newaccount DO
    BEGIN
      oldbalance := newbalance;
      writeln('Customer number: ',custno:4);
      write('Current balance: ',oldbalance:7:2);
      write(' Current payment: ');
      readln(payment);
      IF payment > 0 THEN
        BEGIN
          write('Payment date: ');
          WITH paydate DO
            readln(month,slash,day,slash,year);
          END;
          writeln;
          newbalance := oldbalance - payment;
          IF payment >= 0.1*oldbalance
            THEN custtype := current
            ELSE IF payment > 0 THEN custtype := overdue
                    ELSE custtype := delinquent
          END (* WITH newaccount *)
        END;
      END;
    END;
  END;
```

إذا جمعنا كل الأجزاء الآن مع بعضها ، فيظهر البرنامج الكامل على النحو التالي :

```
PROGRAM billing2(input,output,newfile,oldfile);

(* THIS PROGRAM UPDATES A DATA FILE BY READING AN OLD FILE,
   UPDATING EACH RECORD, AND THEN WRITING THE UPDATED
   RECORDS ONTO A NEW FILE *)

TYPE status = (current,overdue,delinquent);
line = PACKED ARRAY [1..30] OF char;
date = RECORD
  month : 1..12;
  day : 1..31;
  year : 1900..2100
END;
```

(تكملة البرنامج في الصفحة التالية)

```

account = RECORD
    name : line;
    custno : 1..9999;
    custtype : status;
    oldbalance : real;
    newbalance : real;
    payment : real;
    paydate : date
END;
datafile = FILE OF account;
VAR newfile,oldfile : datafile;
    newaccount,oldaccount : account;
PROCEDURE writefile(VAR data : datafile);
(* THIS PROCEDURE CAUSES THE CONTENTS OF A DATA FILE TO BE WRITTEN OUT *)
VAR custaccount : account;
BEGIN
    reset(data);
    WHILE NOT eof(data) DO
        BEGIN
            read(data,custaccount);
            WITH custaccount DO
                BEGIN
                    writeln;
                    write('Name: ',name);
                    writeln(' Customer Number: ',custno:4);
                    writeln;
                    write('Old balance: ',oldbalance:7:2);
                    write(' Current payment: ',payment:7:2);
                    writeln(' New balance: ',newbalance:7:2);
                    write('Payment date: ',paydate.month:2,'/');
                    writeln(paydate.day:2,'/',paydate.year:4);
                    writeln;
                    write('Account status: ');
                    CASE custtype OF
                        current    : writeln('CURRENT');
                        overdue    : writeln('OVERDUE');
                        delinquent : writeln('DELINQUENT')
                    END;
                    writeln
                END (* WITH custaccount *)
            END (* WHILE NOT eof(data) *)
        END; (* writefile *)

PROCEDURE update(VAR newaccount : account);
(* THIS PROCEDURE ENTERS NEW DATA FROM THE KEYBOARD
    AND THEN UPDATES EACH RECORD AS NECESSARY *)
VAR slash : char;
BEGIN
    WITH newaccount DO
        BEGIN
            oldbalance := newbalance;
            writeln('Customer number: ',custno:4);
            write('Current balance: ',oldbalance:7:2);
            write(' Current payment: ');
            readln(payment);

```

(تكملة البرنامج في الصفحة التالية)

```

        IF payment > 0 THEN
            BEGIN
                write('Payment date: ');
                WITH paydate DO
                    readln(month,slash,day,slash,year);
                END;
            writeln;
            newbalance := oldbalance - payment;
            IF payment >= 0.1*oldbalance
                THEN custtype := current
            ELSE IF payment > 0 THEN custtype := overdue
                ELSE custtype := delinquent
            END (* WITH newaccount *)
        END; (* update *)

    BEGIN (* main action block *)
        writeln('CUSTOMER BILLING SYSTEM - FILE UPDATE');
        writeln;
        writeln('OLD DATA FILE');
        writeln;
        writefile(oldfile);

        BEGIN (* update the old file *)
            writeln;
            writeln('DATA UPDATE');
            writeln;
            reset(oldfile);
            rewrite(newfile);
            WHILE NOT eof(oldfile) DO
                BEGIN
                    read(oldfile,oldaccount);
                    newaccount := oldaccount;
                    IF oldaccount.newbalance > 0 THEN update(newaccount);
                    write(newfile,newaccount)
                END (* WHILE NOT eof(oldfile) *)
            END; (* file update *)

            writeln;
            writeln('NEW DATA FILE');
            writeln;
            writefile(newfile)
        END.
    
```

افترض الآن أن البرنامج يستخدم لتشغيل ملف بيانات بسيط يحتوي على ستة حسابات . يبدأ هذا البرنامج بطباعة محتويات ملف البيانات الأصلي oldfile كما يلي :

CUSTOMER BILLING SYSTEM - FILE UPDATE

OLD DATA FILE

Name: Alan B. Adams

Customer Number: 1

Old balance: 128.32 Current Payment: 128.32 New balance: 0.00

Payment date: 11/27/1984

Account Status: CURRENT

(تكملة البرنامج في الصفحة التالية)

Name: James Brown Customer Number: 2  
 Old balance: 189.00 Current Payment: 100.00 New balance: 89.00  
 Payment date: 11/18/1984  
 Account Status: CURRENT

Name: Janet P. Davis Customer Number: 3  
 Old balance: 212.18 Current Payment: 0.00 New balance: 212.18  
 Payment date: 9/ 4/1984  
 Account Status: DELINQUENT

Name: William D. McDonald Customer Number: 4  
 Old balance: 200.00 Current Payment: 65.00 New balance: 135.00  
 Payment date: 11/10/1984  
 Account Status: CURRENT

Name: Phyllis W. Smith Customer Number: 5  
 Old balance: 289.42 Current Payment: 20.00 New balance: 269.42  
 Payment date: 11/21/1984  
 Account Status: OVERDUE

Name: Richard L. Warren Customer Number: 6  
 Old balance: 70.00 Current Payment: 50.00 New balance: 20.00  
 Payment date: 11/15/1984  
 Account Status: CURRENT

بعد كتابة محتويات ملف البيانات القديم ، يلحق البرنامج المستفيد ليُدخل بيانات جديدة لكل عميل لا تكون موازناتها القائمة مساوية للصفر . وعلى هذا ... يجدد خمسة من السجلات . والحوار المتداخل لتجديد البيانات يبدو فى الصورة التالية ( لاحظ أن كل استجابات المستفيد موضوع تحتها خط ) .

## DATA UPDATE

Customer number: 2  
 Current balance: 89.00 Current payment: 50.00  
 Payment date: 12/12/1984

Customer number: 3  
 Current balance: 212.18 Current payment: 212.18  
 Payment date: 12/3/1984

Customer number 4  
 Current balance: 135.00 Current payment: 10.00  
 Payment date: 12/18/1984

Customer number: 5  
 Current balance: 269.42 Current payment: 0

Customer number: 6  
 Current balance: 20.00 Current payment: 20.00  
 Payment date: 12/7/1984

لاحظ أن ملفقات البرنامج لمعلومات جديدة فقط للعملاء الذين تكون موازناتهم الحالية أكبر من الصفر . لاحظ أن البرنامج سوف يلحق أيضا لإدخال تاريخ المبلغ المدفوع إذا كان هناك مبلغ مدفوع أكبر من الصفر فقط .

وبعد إدخال البيانات الجديدة وإتمام التجديد ، يكتب ملف العملاء الجديد . وتظهر محتويات هذا الملف على النحو التالى :

## NEW DATA FILE

Name: Alan B. Adams Customer Number: 1  
 Old balance: 128.32 Current Payment: 128.32 New balance: 0.00  
 Payment date: 11/27/1984  
 Account Status: CURRENT

Name: James Brown Customer Number: 2  
 Old balance: 89.00 Current Payment: 50.00 New balance: 39.00  
 Payment date: 12/12/1984  
 Account Status: CURRENT

Name: Janet P. Davis Customer Number: 3  
 Old balance: 212.18 Current Payment: 212.18 New balance: 0.00  
 Payment date: 9/ 4/1984  
 Account Status: CURRENT

(تكملة البرنامج فى الصفحة التالية)

Name: William D. McDonald Customer Number: 4  
 Old balance: 135.00 Current Payment: 10.00 New balance: 125.00  
 Payment date: 12/18/1984  
 Account Status: OVERDUE

Name: Phyllis W. Smith Customer Number: 5  
 Old balance: 269.42 Current Payment: 0.00 New balance: 269.42  
 Payment date: 11/21/1984  
 Account Status: DELINQUENT

Name: Richard L. Warren Customer Number: 6  
 Old balance: 20.00 Current Payment: 20.00 New balance: 0.00  
 Payment date: 12/ 7/1984  
 Account Status: CURRENT

لاحظ أن أول سجل يظل كما هو بدون أى تغيير ، حيث إن أول عميل ( Alan B. Adams ) ليس لديه موازنة قائمة أثناء فترة التجديد هذه .

وبعد إتمام تنفيذ البرنامج ، يجب أن يحذف oldfile ، ويعاد تسمية newfile باسم oldfile . ويسمح ذلك بتجديد الملف الذى تم تجديده خلال هذه الدورة فى المستقبل ( أو فى الدورة التالية ) .

## 8. TEXT FILES

## ٨ - ملفات النصوص

يمكن استخدام نوع آخر من أنواع الملفات فى البسكال ، وهو ملفات النصوص text files . وهذا الملف من النوع الحرفى ، مع وجود محدد لنهاية السطر فى أماكن مختلفة لتمييز كل سطر من أسطر النص عن بقية سطور النص . ( وعادة ما يحتوى محدد نهاية السطر على تغذية للسطر ، تتبع عودة العربة ) . وعلى هذا ... يتكون ملف النص من أسطر متعددة لبيانات من النوع الحرفى .

وقد يحدث تجميع للرموز الموجودة فى أى سطر ( أى تفصل عن بعضها بواسطة فراغات ) . ويمكن تفسير مجموعة الرموز الفردية بإحدى طرق عديدة . فمثلاً مجموعة من الأرقام الصحيحة يمكن تفسيرها ككمية صحيحة أو حقيقية . وتتحدد طريقة تفسير هذه المجموعات لعبارة read أو عبارة write المصاحبة ( المزيد عن ذلك يذكر فيما بعد ) .

ويمكن تعريف ملف النص بسهولة ، وذلك بإقراره كمعرف ( أى كمتغير يمثل الملف ) من نوع النص . ويمكن أن يعبر رسمياً عن التوضيح كما يلى :

VAR file name : text

حيث file name هو معرف يمثل اسم ملف النص . ويمكن بالطبع وجود عدة أسماء ملفات في توضيح واحد .

مثال (١١-٢١)

افترض أنه يجب على برنامج البسكال أن يستخدم ملفي النص newtext , oldtext . يظهر تعريف الملفين كما يلي :

```
VAR oldtext,newtext : text;
```

لاحظ أنه ليس من الضروري كتابة "FILE OF text" داخل التوضيح . وكل ملف نص له احتياطي ملف مصاحب له ، مثل أى ملف آخر . وحيث إن ملف النص من النوع الحرفي ، فإن احتياطي ملف النص يمثل رمزا واحدا . ويمكن استخدام الإجراءات القياسية reset , rewrite مع ملفات النصوص ، كما يستخدمان تماما مع الأنواع الأخرى من الملفات . وتستخدم عبارة reset لإعداد الملف للقراءة . وعلى هذا ... فإن العبارة التالية :

```
reset(file name);
```

تتسبب في تحديد أول رمز لاحتياطي الملف ، على ألا يكون الملف فارغا . والدالة القياسية :

```
eof(file name)
```

تضع قيمة مناسبة عند إعداد الملف .

وبالمثل تعد عبارة rewrite للكتابة في الملف . وعلى هذا ... فإن العبارة :

```
rewrite(file name);
```

تمسح النص الموجود فعلا في ملف النص ( إذا ما كان موجودا ملف ) ، ثم تحدد بداية ملف النص .

كما يمكن استخدام الإجراءات القياسية أيضا مع ملفات النصوص ، مثل استخدامهما مع أنواع الملفات الأخرى تماما . وعلى هذا ... فالعبارة

```
get(file name);
```

تتسبب في تحديد الرمز التالي في ملف النص لاحتياطي الملف . ( في العادة يكون قد تم تشغيل الرمز السابق قبل قراءة الرمز التالي ) .

```
put(file name);
```

أيضا في كتابة الرمز الموجود حاليا في احتياطي الملف في الملف . وكقاعدة عامة ... يسبق عبارة put عبارة تحديد ، وذلك لوضع رمز جديد في احتياطي الملف . وسوف يكتب هذا الرمز الجديد في ملف النص كآخر لعبارة put .

وتستخدم معظم البرامج التي تستخدم ملفات نصوص الإجراءات القياسية read , readln , write , writeln . بدلا من put , get . دعنا على ذلك نعيد فحص هذه الإجراءات في ضوء مناقشتنا الحالية للملفات النصوص .

إذا ما استخدمت العبارة التالية :

```
read(file name, variable name);
```

مع ملف نص ، فيعتمد الإجراء المأخوذ على نوع بيانات المتغير .

إذا كان المتغير من النوع الحرفي ، فيتم تشغيل عنصر البيانات بالطريقة المعتادة ، أى يتحدد الرمز الموجود فى احتياطي الملف للمتغير ، يتبعه قراءة رمز جديد من ملف النص ، وتحديد احتياطي الملف . أما إذا كان المتغير من النوع الصحيح أو النوع الحقيقي ، فيقرأ عدداً كافياً متتالياً من الرموز ليسمح بتكوين كمية صحيحة أو حقيقية . عند ذلك يتم تحديد هذه الكمية للمتغير ، ثم يتم تحديد الرمز التالى فى الملف لاحتياطي الملف .

ويمكن أن تظهر أسماء متغيرات متعددة فى عبارة read واحدة . وتفسر مثل هذه العبارة كتسلسل من عبارات read ، كل منها يحتوى على اسم متغير واحد . وعلى هذا ... فالعبارة :

```
read(file name, variable name 1, variable name 2, . . . , variable name n);
```

تفسر كما يلى :

```
BEGIN
  read(file name, variable name 1);
  read(file name, variable name 2);
  .
  .
  .
  read(file name, variable name n)
END;
```

إذا مامتلت المتغيرات كميات عديدة متتالية ، فعلى ذلك يعتمد فصل كميه عديدة عن كمية أخرى على اكتشاف الفراغات التى تسبقها ، أو اكتشاف رموز سطر التحكم كما سبق ذكره فى الفصل الرابع ( انظر القسم ٢ من الفصل الرابع ) .

وتشبه عبارة readln عبارة read . فى هذه الحالة تتسبب عبارة readln واحدة فى استمرار القراءة حتى يظهر المحدد الخاص بنهاية السطر ( eoln ) . وعلى هذا ... فإن أى read تالية أو readln تالية تبدأ فى السطر التالى ( انظر القسم ٣ من الفصل الرابع ) .

ولنكون محددين أكثر ، فإن العبارة :

```
readln(file name);
```

تكافئ .

```
BEGIN
  WHILE NOT eoln(file name) DO get(file name);
  get (file name)
END;
```

( لاحظ أن اخر get تتسبب فى تحديد أول رمز فى السطر التالى لاحتياطي الملف ) . وأكثر من هذا ... العبارة :

```
readln(file name, variable name 1, variable name 2, . . . , variable name n);
```



تفسر كما يلي :

```
BEGIN
  read(file name, variable name 1, variable name 2, . . . , 'variable name n');
  readln(file name)
END;
```

والعبارة القياسية :

```
coln(file name)
```

هي دالة بوليانية قياسية تعود بقيمة خاطئة false ، إلا إذا كان الرمز الحالي في الملف هو مؤشر نهاية الملف . وعلى هذا ... تظل الدالة coln خاطئة false حتى تكتشف نهاية الملف ، والتي تسبب في جعلها صحيحة true . وبعد ذلك تصبح الدالة خاطئة false مرة أخرى عندما يفحص رمز آخر بعد اكتشاف نهاية السطر ( انظر قسم ٤ من الفصل الرابع ، وقسم ١ من الفصل التاسع ) .

دعنا الآن نوجه انتباهنا إلى عبارات write and writeln

إذا استخدمت العبارة .

```
write(file name, output item)
```

مع ملف ، فيعتمد الإجراء المتخذ على طبيعة عنصر المخرجات . إذا كان عنصر المخرجات من النوع الحرفي ( أى أنه ثابت أو متغير من النوع الحرفي ) ، فعند ذلك يتحدد الرمز لاحتياطي الملف ، ويكتب في الملف . أما إذا مامثل عنصر المخرجات نوع بيانات آخر ( أى متغير بولياني ، أو ثابت عددي ، أو متغير عددي ، أو تعبير عددي ) فعند ذلك تتحول قيمة عنصر المخرجات إلى الرموز التي تحتويها . ونكتب هذه الرموز بعد ذلك تلقائياً في ملف النص .

ويمكن أن تظهر عناصر مخرجات متعددة في عبارة write واحدة . وتفسر مثل هذه العبارة على أنها تسلسل من عبارات write ، كل منها يحتوي على عنصر مخرجات واحد . وعلى هذا ... فإن العبارة .

```
write(file name, output item 1, output item 2, . . . , output item n);
```

تفسر على أنها :

```
BEGIN
  write(file name, output item 1);
  write(file name, output item 2);
  .
  .
  .
  write(file name, output item n)
END;
```

انظر القسم ( ٥ من الفصل الرابع )

تتسبب العبارة .

```
writeln(file name);
```

في كتابة تحديد لنهاية سطر في الملف . وعلى هذا ... فأى مخرجات تالية لملف النص تبدأ على السطر التالي . وأكثر من ذلك ... العبارة

```
writeln(text file, output item 1, output item 2, . . . , output item n);
```

تفسر كما يلي :

```
BEGIN
```

```
  write(file name, output item 1, output item 2, . . . , output item n);  
  writeln(file name)
```

```
END;
```

وتسمح بذلك لعناصر مخرجات متعددة أن توضع على نفس السطر ، يتبعها تحديد لنهاية السطر ( انظر القسم ٦ من الفصل الرابع ) .

وأخيرا يمكن استخدام الإجراء القياسي page مع أى ملف من ملفات النصوص ، وذلك بوضع أسم الملف كمؤشر أى :

```
page(file name);
```

ويتسبب استخدام هذا الإجراء في بدء كل مخرج جديد في بداية صفحة جديدة ، على أن يكون هذا المخرج موجها لوحدة يمكنها تمييز بداية الصفحة ( أى طابع أسطر ) ، وإلا فسوف تهمل عبارة page ( انظر القسم ٧ من الفصل الرابع ) .

مثال (١١-٢٢)

نقل النصوص بين ملفات النصوص . فيما يلي برنامج بسكال بسيطاً ينقل محتويات oldtext إلى newtext سطراً سطراً

( لاحظ أن كلا من oldtext و newtext هو ملف نص ) .

```
PROGRAM filetransfer(oldtext,newtext);  
(* THIS PROGRAM TRANSFERS TEXT FROM ONE TEXT FILE  
   TO ANOTHER ON A LINE-BY-LINE BASIS *)  
VAR oldtext,newtext : text;  
    x : char;  
BEGIN  
  reset(oldtext);  
  rewrite(newtext);  
  page(newtext);  
  WHILE NOT eof(oldtext) DO  
    BEGIN  
      WHILE NOT eoln(oldtext) DO  
        BEGIN (* transfer one line of text *)  
          read(oldtext,x);  
          write(newtext,x)  
        END;  
        readln(oldtext);  
        writeln(newtext)  
      END  
    END  
END.
```

تتسبب دورة DO - WHILE الداخلية في قراءة النص الموجود في سطر من أسطر oldtext وكتابه في newtext. ويحدث النقل رمزاً رمزاً ، وذلك بالنسبة لأي سطر من أسطر النص . يلي دورة DO - WHILE عبارة read التي تعد البرنامج لقراءة السطر الثاني من oldtext ، ثم عبارة writeln التي تضع محدد نهاية السطر في السطر الحالي في newtext .

وتتسبب دورة DO - WHILE الخارجية في تكرار العمل سطراً سطراً ، حتى يكتشف محدد نهاية الملف في oldtext .

ويوجد ملفان سبق توضيحهما في البسكال ، هما : input , output ، ويستخدمان في نقل بيانات مدخلات ومخرجات من وإلى وحدات مدخلات ومخرجات قياسية . وقد سبق لنا بالطبع استخدام هذه الملفات منذ ظهورهما في الفصل الرابع من الكتاب ، بالرغم من أننا لم نعرفهما بأنهما ملفا نصوص حتى هذه النقطة .

وعند استخدام إجراءات أو دوال قياسية سبق ذكرها مع أحد هذين الملفين الذي سبق توضيحهما ، فلا يكون من الضروري أن يظهر اسم الملف كمؤشر ، فإذا لم يتم تحديد المؤشر ، فيفترض input أو output طبقاً للإجراء الخاص أو الدالة الخاصة . وعلى هذا ... فالإجراءات والدوال التالية يفترض أنها مطبقة على ملف نص output إذا لم يوجد مؤشر ملف صريح : write , writeln , page . أكثر من ذلك ... تعد input , output تلقائياً ، بحيث إن الإجراءين القياسيين rewrite , reset لا يكون هناك حاجة لهما مع هذه الملفات .

### مثال (١١-٢٣)

إدخال وحفظ ملف نص . فيما يلي برنامجاً يسمح بإدخال العديد من أسطر أحد النصوص من وحدة مدخلات ، وتخزينها في ملف نص اسمه newtext . ( لاحظ أن هذا البرنامج صيغة أخرى من البرنامج الموجود في مثال ١١ - ٢٢ ) .

```
PROGRAM entertext(input,newtext);
(* THIS PROGRAM ACCEPTS TEXT FROM AN INPUT DEVICE AND
   STORES IT IN A TEXT FILE ON A LINE-BY-LINE BASIS *)
VAR newtext : text;
    x : char;
BEGIN
    rewrite(newtext);
    WHILE NOT eof DO
        BEGIN
            WHILE NOT eoln DO
                BEGIN (* enter and store one line of text *)
                    read(x);
                    writs(newtext,x)
                END;
            readln;
            writeln(newtext)
        END
    END.
```

عند تنفيذ هذا البرنامج ، يمكن إدخال عدد غير محدد من الأسطر من وحدة المدخلات ، وتخزينها فى newtext . والفصل التنفيذ يجب إدخال محدد لنهاية الملف من وحدة المدخلات . وتختلف طبيعة هذا المحدد من منشأه لمنشأه أخرى ، بالرغم من أن رموز التحكم يتكرر استخدامها فى هذا الغرض .

### مثال (١١-٢٤)

تشفير وإعادة تشفير النص . نعتبر الآن تطبيقاً أكثر أهمية ، يشمل استخدام ملفات نصوص . دعنا نطور برنامجاً متداخلاً بلغة البسكال ، يعمل شفرة لعدة أسطر من أسطر أحد النصوص ، ثم يفك هذه الشفرة . ويحتوى البرنامج على قائمة تسمح باتخاذ أى إجراء من الإجراءات التالية :

١ - إدخال النص بواسطة لوحة المفاتيح ، وعمل شفرة للنص ، وتخزين النص وهو فى حالة هذه فى ملف نص .

٢ - استرجاع الملف المسجل وهو فى حالة الشفرة ، وعرضه فى صورته هذه .

٣ - استرجاع الملف المسجل وهو فى حالة الشفرة ، وفك الشفرة ، ثم عرضه بعد فك الشفرة .

٤ - إنهاء الحسابات .

يمكن أن يحتوى جسم النص على العديد من الأسطر . وسوف تحفظ هذه الأسطر عندما يتم عمل شفرة للنص ، ثم يخزن ، وبعد ذلك تفك الشفرة ويعرض النص . وعلى أية حال ... فيحدث عمل الشفرة وفكها رمزا رمزا .

والكى يتم عمل شفرة وفكها للنص ، يجب أن يدخل المستخدم رمزا فرديا "Key" ( سوف يلقي البرنامج لعمل ذلك إذا ما أختير عنصر القائمة 1 أو 3 ) . ويتم عمل شفرة لكل رمز وذلك بإضافة شفرته العددية للشفرة العددية للمفتاح key ثم تحديد الرمز الذى يمثل هذا المجموع ، أى :

$$z := \text{chr}(\text{ord}(x) + \text{ord}(\text{key}))$$

حيث z تمثل شفرة الرمز المكافئ للرمز الأصلي x .

وعند إضافة الشفرتين العدديتين يجب أن نكون حريصين ، حتى لا يتعدى مجموعهما ١٢٧ ، حيث إن ١٢٧ هى أكبر قيمة لشفرة رمز قياسى ( يفرض مجموعة رموز ASCII ) . وعلى هذا ... إذا ماتعدت قيمة z العدد ١٢٧ ، فيجب أن تضبط قيمتها بطرح ١٢٧ من المجموع . وعلى هذا ... يمكننا كتابة :

```
y := ord(x) + ord(key);
IF y > 127 THEN y := y - 127;
z := chr(y);
```

تعمل هذه الطريقة جيداً وهى سهلة التنفيذ ، ويجب أن يكون مفهوماً على أية حال أن بعض قيم z تنتج عنها رموز غير قابلة للطباعة ( أى تغذية أسطر ، أو عودة العربة ، أو beeps .. الخ ) . وسوف توضع هذه الرموز بالطبع فى النص المكتوب بالشفرة . وعلى هذا ... فيمكن أن يظهر النص المكتوب بالشفرة فى شكل غريب عند عرضه .

ويمكن إعادة فك الشفرة ، وذلك بعكس العملية السابقة ، أى :

```
y := ord(z) - ord(key);
IF y < 0 THEN y := y + 127;
x := chr(y);
```

وعند فك شفرة النص يكون من الضرورى بالطبع أن نستخدم نفس قيمة المفتاح key التى سبق استخدامها عند عمل الشفرة للنص الأسمى ، وإلا فإن فك الشفرة سوف يؤدى إلى نص غريب .

وفيما يلى تخطيطا هيكليا لمحتويات برنامج يوضح طريقة الحسابات الشاملة .

```
PROGRAM encode(input,output,code);
TYPE features = 1..4;
VAR code : text;
    x,z,key : char;
    y : -127..254;
    choice : features;
PROCEDURE menu(VAR choice : features; VAR key : char);
BEGIN
    .
    .
    (* generate a menu, return the user's choice
       and a value for the key, if appropriate *)
    .
    .
END;
BEGIN (* main action statements *)
    choice := 1;
    WHILE choice <> 4 DO
        menu(choice,key);
        CASE choice OF
            1 : BEGIN
                .
                .
                (* enter text, encode and store *)
                .
                .
            END;
            2,3 : BEGIN
                .
                .
                (* retrieve encoded text, decode
                   (if choice = 2) and display *)
                .
                .
            END;
            4 : ;
        END (* CASE *)
    END (* WHILE loop *)
END.
```

والطريقة الشاملة مباشرة ، وتعتمد على استخدام مكون CASE في تنفيذ الاختيار المطلوب من القائمة . وبعد إتمام كل نشاط أساسي ( بعد كل ممر خلال مكون CASE ) ، يعود البرنامج إلى القائمة لأداء اختيار آخر ، حتى يتم اختيار البديل رقم ٤ .

وفيما يلي برنامج البسكال كاملاً :

```
PROGRAM encode(input,output,code);

(* THIS IS A MENU-DRIVEN PROGRAM THAT ACCEPTS TEXT FROM AN INPUT DEVICE,
   ENCODES IT, STORES THE ENCODED TEXT IN A TEXT FILE, AND DISPLAYS THE
   TEXT, EITHER ENCODED OR DECODED *)

TYPE features = 1..4;
VAR code : text;
    x,z,key : char;
    y : -127..254;
    choice : features;

PROCEDURE menu (VAR choice : features; VAR key : char);

(* This procedure generates a menu and returns the user's choice *)

BEGIN
    writeln;
    writeln('E N C O D I N G / D E C O D I N G   T E X T');
    writeln;
    writeln('Program Features:');
    writeln;
    writeln('  1 - Enter text, encode and store');
    writeln;
    writeln('  2 - Retrieve encoded text and display');
    writeln;
    writeln('  3 - Retrieve encoded text, decode and display');
    writeln;
    writeln('  4 - End computation');
    writeln;
    write('Please enter your selection (1, 2, 3 or 4) -> ');
    readln(choice);
    writeln;
    IF (choice = 1) OR (choice = 3) THEN
        BEGIN
            write('Please enter the key (one character) -> ');
            readln(key);
            writeln
        END
    END;

(* menu *)

BEGIN (* main action statements *)
    choice := 1;
    WHILE choice <> 4 DO
        BEGIN
            menu(choice,key);
            CASE choice OF
```

(تكملة البرنامج في الصفحة التالية)

```

1 : BEGIN (* enter text, encode and store *)
    rewrite(code);
    writeln('Enter text:');
    writeln;
    WHILE NOT eof DO
        BEGIN
            WHILE NOT eoln DO
                BEGIN
                    read(x);
                    y := ord(x) + ord(key);
                    IF y > 127 THEN y := y - 127;
                    z := chr(y);
                    write(code,z)
                END;
            readln;
            writeln(code)
        END; (* NOT eof *)
        reset(input) (* eof=false *)
    END;

2,3 : BEGIN (* retrieve encoded text, decode and display *)
    reset(code);
    WHILE NOT eof(code) DO
        BEGIN
            WHILE NOT eoln(code) DO
                BEGIN
                    read(code,z);
                    IF choice = 2
                        THEN x := z
                        ELSE BEGIN
                            y := ord(z) - ord(key);
                            IF y < 0 THEN y := y + 127;
                            x := chr(y)
                        END;
                    write(x)
                END; (* NOT eoln *)
            readln(code);
            writeln
        END (* NOT eof *)
    END;

4 : ;

END (* CASE *)
END; (* WHILE choice <> 4 *)
writeln;
writeln('That's all, folks!')
END.

```

يقدم الإجراء menu عبارات المدخلات والمدخلات اللازمة لإنتاج القائمة ، وإذا ما اختار المستخدم الاختيار ١ أو ٢ ، يلحق لاختيار مفتاح key رمزا واحدا . لاحظ أن هذا الإجراء يستخدم مؤشرين متغيرين هما : choice ، وهو من نوع key و features وهو من النوع الحرفي .

وتحتوى المجموعة الأساسية أساسا على مكون CASE به أربعة اختيارات مختلفة . أول اختيار يناظر Choice = 1 ، ويقدم التعليمات اللازمة لإدخال أسطر متعددة من البيانات وعمل الشفرة ، ثم التخزين سطرًا سطرًا . لاحظ أن الرمز الذى أجرى لها الشفرة كُتِبَ فى ملف النص Code .

ويستمر إدخال البيانات حتى يتم إدخال مايفيد انتهاء الملف من لوحة المفاتيح . ( وطريقة تحقيق ذلك تتغير من جهاز كمبيوتر لجهاز آخر ، بالرغم من أنه عادة ما يستخدم رمز تحكم فى هذا الغرض ) . وبعد إتمام إدخال البيانات ، يعاد إعداد reset ملف النص input ، بحيث إن شرط eof يكون خاطئا false مرة أخرى .

فإذا كانت قيمة Choice هي ٢ أو ٣ ، فإن الملف Code يعاد إعداد reset ، ويقرأ النص المشفر رمزا رمزا . إذا كان Code = 2 يكتب النص المشفر مباشرة فى وحدة المخرجات .

وإلا ( إذا كان Code = 3 ) تفك شفرة الرمز ، ثم تكتب فى وحدة المخرجات .

وعند تنفيذ البرنامج تنتج القائمة التالية .:

#### ENCODING / DECODING TEXT

##### Program Features:

- 1 - Enter text, encode and store
- 2 - Retrieve encoded text and display
- 3 - Retrieve encoded text, decode and display
- 4 - End computation

Please enter your selection (1, 2, 3 or 4) ->

إذا ما أدخلت القيمة ١ أو ٢ كاختيار ، فينتج الملقن التالى للمفتاح key .

Please enter the key (one character) ->

افرض الآن أن المستفيد اختار إدخال عدة أسطر فى الملف . يجب على المستفيد أن يدخل القيمة ١ أولا للاختيار . افرض أن المستفيد أدخل الرمز C للمفتاح key ، وتبعه النص .:

All digital computers, regardless of their size, are basically electronic devices that can transmit, store and manipulate information (i.e., data).  
^Z

( الرمز ^Z فى نهاية النص تحدد نهاية الملف ) . يتم عمل شفرة لهذه المعلومات رمزا رمزا ، ثم تخزن فى ملف النص Code . ويتبع ذلك إعادة ظهور القائمة .

إذا ما اختار المستفيد الآن الاختيار الثانى ، فقرأ البيانات المشفرة من ملف النص Code ، وتعرض :

```
00c(++-8%0c'31498)67oc6)+%6(0)77c3+c8,)-6c7->)oc%6)
&%7-'%00=c)0)'8632-'c():-'')7c8,%8c'%2c86%271-8oc7836)c%2(
1%2-490%8)c-2+361%8-32ck-q)qoc(%8%1q
```



وهذه بالطبع هى الصيغة المشفرة للنص الذى سبق إدخاله .

افرض الآن أن المستخدم اختار الاختيار الثالث . يظهر ملقن المفتاح Key مرة أخرى . ويجب أن يستجيب المستخدم بإدخال C ، وهو نفس المفتاح الذى استخدمه فى عمل الشفرة للنص . وبمجرد إدخال المفتاح تقرأ البيانات المشفرة مرة أخرى من ملف النص Code ، وتلك شفرتها رمزا رمزا ، ثم تعرض على النحو التالى :

All digital computers, regardless of their size, are  
basically electronic devices that can transmit, store and  
manipulate information (i.e., data).

وعلى هذا ... فقد أعيد تكوين النص الأصيل .

إذا ما استخدم مفتاح خاطئ وتم إدخاله كاستجابة للملقن ، فيعاد فك الشفرة للنص بطريقة خاطئة ، وتضيع معالم النص الأصيل ، ويظهر نص لامعنى له . افرض على سبيل المثال أن المستخدم أدخل B ، بدلا من C كمفتاح Key . عند ذلك يظهر النص التالى :

Bmmlejhzubm!dpnqvufst-!sfhbsmftt!pg!uifjs!tj(f-lbsf  
cbtjdbmmz!fmfduspojdl!efwjdf!uibu!dbo!usbotn!ju-!tupsf!boe  
nbojqvmbuf!jogpsnbujpo!))z/f/-!ebub\*/

أخيرا ، افرض أن المستخدم اختار إنهاء الحسابات باختياره للبديل الأخير فى القائمة . عند ذلك تظهر الرسالة التالية :

That's all, folks!

وتفصل الحسابات .

## Review Questions

## أسئلة مراجعة :

- (١) ماهى الخواص الأساسية للملف ؟ وكيف يختلف الملف عن المنظومة ؟
- (٢) ماهو الفرق بين الملف الدائم والملف المؤقت ؟ ماهو الاصطلاح المستخدم للإشارة إلى الملفات الدائمة فى البسكال ؟
- (٣) ماهو الفرق بين الملف الخارجى والملف الداخلى ؟
- (٤) ماهو الفرق بين الملف التتابعى والملف العشوائى ؟ وما هى مميزات وعيوب كل منهما ؟
- (٥) ماهو الاسم الآخر للملف الاتصال العشوائى ؟ أى الاسمين معبر أكثر ؟ وضح ذلك .
- (٦) ماهو نوع الملفات المستخدم فى بسكال ISO القياسى ؟
- (٧) لخص قواعد تعريف نوع الملفات .

- (٨) لخص قواعد تعريف متغير من نوع ملف . قارن ذلك بإجابتك على السؤال السابق .
- (٩) ماهى أنواع البيانات التى يمكن أن تصاحب مكونات ملف فردى ؟
- (١٠) ماهى القيود الموضوعة على أقصى طول مسموح به للملف ؟ وهل هذه القيود موضوعة على أى نوع من أنواع البيانات المرتبة الأخرى ؟
- (١١) كيف تمرر الملفات الخارجية إلى أو من برنامج البسكال ؟ وكيف يتحقق ذلك ؟
- (١٢) ماهو احتياطى الملف ؟ وكيف يسمى احتياطى الملف ؟ وما نوع البيانات التى يمكن أن يمثلها ؟
- (١٣) ماهى أول خطوة فى إنتاج ملف جديد ؟ وكيف يمكن تحقيق ذلك ؟
- (١٤) ماهو تأثير عبارة rewrite عند إعداد الملف الموجود للكتابة ؟
- (١٥) ماهو الغرض من عبارة put ؟ وكيف يمكن تكرار استخدام هذه العبارة فى كتابة مكونات فى ملف ؟
- (١٦) افرض أن أحد الملفات يحتوى على مكونات مرتبة يراد إنتاجه . كيف يمكن إدخال عناصر البيانات الفردية لكل مكون من مكونات الملف داخل الكمبيوتر وكتابتها فى الملف ؟ هل يجب أن تكون كل عناصر البيانات هذه من نفس النوع ؟
- (١٧) كيف يمكن تمرير إحدى مكونات ملف مرتب كمؤشر بين إجراء والمجموعة الإجرائية الرئيسية ؟
- (١٨) ماهو الغرض من عبارة write ؟ وكيف تختلف هذه العبارة عن عبارة put ؟ وضح ذلك بالتفصيل .
- (١٩) ما نوع عناصر المخرجات التى يمكن أن توجد فى عبارة write ؟ هل يمكن وجود عناصر مخرجات متعددة فى عبارة write واحدة ؟
- (٢٠) كيف يمكن إعداد ملف موجود للقراءة ؟
- (٢١) ماهو تأثير عبارة reset على ملف يحتوى على مكون واحد أو أكثر ؟
- (٢٢) ماهو تأثير عبارة reset على ملف فارغ ؟ قارن إجابتك بالإجابة المعطاه للسؤال السابق .
- (٢٣) ماهو الغرض من عبارة get ؟ وكيف يمكن تكرار استخدام هذه العبارة فى قراءة مكونات متعددة من ملف ؟
- (٢٤) متى يقرأ الملف ؟ وكيف تكتشف نهايته ؟
- (٢٥) ماهو الغرض من عبارة read ؟ وكيف تختلف هذه العبارة عن عبارة get ؟ وضح ذلك بالتفصيل .
- (٢٦) ما نوع عناصر المدخلات التى يمكن أن تظهر فى عبارة read ؟ هل يمكن أن تظهر متغيرات متعددة فى عبارة read ؟

- (٢٧) هل يمكن لبرنامج بسكال أن يقرأ معلومات من ملف ، ثم يضيف معلومات جديدة لنفس الملف ؟ وضح ذلك .
- (٢٨) لخص الخطوات التي يجب اتباعها لتجديد ملف موجود .
- (٢٩) هل يمكن نقل ملف إلى إجراء أو إلى دالة كمؤشر قيمة ؟ وضح ذلك .
- (٣٠) ماهو ملف النص ؟ وكيف تختلف ملفات النصوص عن أنواع الملفات الأخرى ؟
- (٣١) كيف تفسر الرموز الموجودة في أحد أسطر أحد النصوص ؟
- (٣٢) كيف يعرف ملف النص ؟
- (٣٣) هل يمكن استخدام عبارة get وعبارة put مع ملف النص ؟
- (٣٤) هل تستخدم عبارة get وعبارة put بصورة معتادة مع ملف النص ؟ وضح ذلك .
- (٣٥) هل يمكن استخدام الإجراءات القياسية الموجهة للملفات ، مثل eof , rewrite , reset مع ملف النص ؟
- (٣٦) ماهو الغرض من عبارة readln ؟ وكيف تختلف هذه العبارة عن عبارة read ؟ هل يمكن استخدام عبارة readln مع كل أنواع الملفات ؟
- (٣٧) افرض أن عبارة read , readln تحتوي على متغير من النوع الصحيح أو الحقيقي . كيف تفسر الرموز المناظرة في ملف النص ؟
- (٣٨) ماهو الغرض من الدالة القياسية eoln ؟ قارن ذلك بالدالة القياسية eof .
- (٣٩) ماهو الغرض من عبارة writeln ؟ وكيف تختلف هذه العبارة عن عبارة write ؟ هل يمكن استخدام عبارة writeln مع كل أنواع الملفات ؟
- (٤٠) افرض أن عبارة write أو عبارة writeln تحتوي على متغير أو على تعبير من نوع بسيط ، بدلا من النوع الحرفي . كيف تفسر الرموز المناظرة في ملف النص ؟
- (٤١) ماهو الغرض من الإجراء القياسي page ؟ هل يفسر هذا الإجراء بنفس الطريقة بواسطة كل وحدات المخرجات ؟
- (٤٢) اعمل تخطيطا لطريقة نقل محتويات ملف نص من أحد ملفات النصوص إلى ملف نص آخر سطرا سطرا .
- (٤٣) كيف يختلف الملفان سابقى التوضيح input , output عن ملفات النصوص الأخرى ؟
- (٤٤) صف التبسيطات المسموح بها عند استخدام أحد ملفات النصوص سابقة التوضيح مع إجراءات ودوال قياسية ، مثل eof , write , read , page .
- (٤٥) هل عبارة reset مطلوبة عند الإعداد لقراءة ملف مدخلات ، وهل rewrite مطلوبة مع ملف مخرجات ؟

## Solved Problems

## مسائل محلولة :

(٤٦) فيما يلي تعريفات عديدة للملفات

- (a) VAR data : FILE OF integer;
- (b) VAR sales, costs : FILE OF real;
- (c) TYPE acctno = 1..9999;  
VAR accounts : FILE OF acctno;
- (d) TYPE color = (red, green, blue);  
sample = RECORD  
    first : color;  
    second : 1..132;  
    third : char  
END;  
VAR data : FILE OF sample;
- (e) TYPE color = (red, green, blue);  
sample = RECORD  
    first : color;  
    second : 1..132;  
    third : char  
END;  
list = ARRAY [1..100] OF sample;  
VAR data : FILE OF list;
- (f) VAR oldstuff, newstuff : text;

(٤٧) توضح البرامج التخصيصية الآتية عمليات تقليدية للملفات :

- (a) Enter a sequence of records from the keyboard and save in a file.

```
PROGRAM sample(input, data);
TYPE line = PACKED ARRAY [1..80] OF char;
personal = RECORD
    name : line;
    address : line;
    phone : line;
END;
VAR data : FILE OF personal;
nameandaddress : personal;
PROCEDURE readline(VAR info : line);
VAR count : 1..80;
BEGIN
    FOR count := 1 TO 80 DO read(info[count]);
    readln
END;
```

```

BEGIN
    rewrite(data);
    WITH nameandaddress DO
        BEGIN
            readline(name);
            WHILE (name[1] <> 'e') AND (name[2] <> 'n')
                AND (name[3] <> 'd') DO
                BEGIN
                    readline(address);
                    readline(phone);
                    write(data,nameandaddress);
                    readline(name)
                END
            END
        END
    END.

```

- (b) Read a sequence of records from a file and display.

```

PROGRAM sample(output,data);
TYPE line = PACKED ARRAY [1..80] OF char;
    personal = RECORD
        name : line;
        address : line;
        phone : line
    END;
VAR data : FILE OF personal;
    nameandaddress : personal;
BEGIN
    reset(data);
    WITH nameandaddress DO
        WHILE NOT eof(data) DO
            BEGIN
                read(data,nameandaddress);
                writeln(name);
                writeln(address);
                writeln(phone);
                writeln
            END
        END
    END.

```

- (c) Copy a sequence of records from one file to another.

```

PROGRAM sample(data1,data2);
TYPE line = PACKED ARRAY [1..80] OF char;
    personal = RECORD
        name : line;
        address : line;
        phone : line
    END;
VAR data1,data2 : FILE OF personal;
    nameandaddress : personal;
BEGIN
    reset(data1);
    rewrite(data2);
    WHILE NOT eof(data1) DO
        BEGIN
            read(data1,nameandaddress);
            write(data2,nameandaddress)
        END
    END
END.

```

- (d) Copy a sequence of records from one file to another, using get and put.

```
PROGRAM sample(data1,data2);
TYPE line = PACKED ARRAY [1..80] OF char;
  personal = RECORD
    name : line;
    address : line;
    phone : line
  END;
VAR data1,data2 : FILE OF personal;
    nameandaddress : personal;

BEGIN
  reset(data1);
  rewrite(data2);
  WHILE NOT eof(data1) DO
    BEGIN
      nameandaddress := data1↑;
      get(data1);
      data2↑ := nameandaddress;
      put(data2)
    - END
  END.
```

- (e) Enter a name from the keyboard. Then search a file for a record with the same name.

```
PROGRAM sample(input,data);
TYPE line = PACKED ARRAY [1..80] OF char;
  personal = RECORD
    name : line;
    address : line;
    phone : line
  END;
VAR count : 1..80;
    newname : line;
    data : FILE OF personal;
    nameandaddress : personal;

BEGIN
  reset(data);
  FOR count := 1 TO 80 DO read(newname[count]);
  REPEAT
    read(data,nameandaddress)
  UNTIL nameandaddress.name = newname
END.
```

- (f) Enter a sequence of names, addresses and phone numbers and save in a text file.

```
PROGRAM sample(input,data);
TYPE line = PACKED ARRAY [1..80] OF char;
VAR name,address,phone : line;
    data : text;

PROCEDURE readline(VAR info : line);
VAR count : 1..80;
BEGIN
  FOR count := 1 TO 80 DO read(info[count]);
  readln
END;
```

```
BEGIN
  rewrite(data);
  readline(name);
  WHILE (name[1] <> 'e') AND (name[2] <> 'n')
    AND (name[3] <> 'd') DO
    BEGIN
      readline(address);
      readline(phone);
      write(data,name,address);
      writeln(data,phone);
      readline(name)
    END
  END.
```

## Supplementary Problems

## مشاكل متكاملة :

(٤٨) التخطيطات الهيكلية التالية توضح حالات عديدة مختلفة ، تشمل استخدام ملفات ، وبعضها مكتوب بطريقة خاطئة . حدد كل الأخطاء .

```
(a) TYPE olddata,newdata : FILE OF integer;
(b) VAR flags : FILE OF boolean;
(c) VAR story : FILE OF text;
(d) TYPE status = (current,overdue,delinquent);
    account = RECORD
        custname : PACKED ARRAY [1..80] OF char;
        custno : 1..9999;
        custtype : status;
        custbalance : real
    END;
    listing = ARRAY [1..1000] OF account;
    posting = FILE OF listing;
    VAR oldposting,newposting : posting;
(e) TYPE color = (red,green,blue);
    sample = RECORD
        first : color;
        second : 1..132;
        third : char
    END;
    data = FILE OF sample;
    VAR olddata,newdata : FILE OF data;
(f) VAR input,output : text;
(g) PROGRAM sample(input,data);
    VAR data : FILE OF real;
    item : real;
    BEGIN
        readln(item);
        WHILE item <> 0 DO
            BEGIN
                writeln(data,item);
                readln(item)
            END
        END.
(h) PROGRAM sample(file1,file2);
    VAR file1,file2 : text;
    byte : char;
    BEGIN
        reset(file2);
        rewrite(file1);
        WHILE NOT eof(file2) DO
            BEGIN
                WHILE NOT eoln(file2) DO
                    BEGIN
                        read(file2,byte);
                        write(file1,byte)
                    END;
                readln(file2);
                writeln(file1)
            END
        END.
END.
```



```

(i) PROGRAM sample(input,output,posting);
    TYPE line = PACKED ARRAY [1..80] OF char;
        account = RECORD
            custname : line;
            custno : 1..9999;
            custbalance : real
        END;
    VAR name : line;
        count : 1..80;
        customer : account;
        posting : FILE OF account;
    BEGIN
        reset(posting);
        customer := posting;
        FOR count := 1 TO 80 DO read(name[count]);
        WHILE customer.custname <> name DO
            BEGIN
                get(posting);
                customer := posting
            END;
        WITH customer DO
            writeln(custname,custno,custbalance)
        END.
(j) PROGRAM sample(input,output,oldaccts,newaccts);
    TYPE line = PACKED ARRAY [1..80] OF char;
        account = RECORD
            custname : line;
            custno : 1..9999;
            custbalance : real
        END;
        filetype = FILE OF account;
    VAR oldaccts,newaccts : filetype;
    PROCEDURE rearrange(oldfile,newfile : filetype);
    BEGIN
        .
        .
        (* rearrange the records by ascending customer number *);
        .
        .
    END;

    BEGIN (* main action statements *)
        .
        .
        rearrange(oldaccts,newaccts);
        .
        .
    END.

```

## Programming Problems

### مشاكل برمجة :

(٤٩) اكتب برنامجا كاملا بلغة البسكال ، ينتج ملف البيانات القديم الموضح فى مثال ١١ - ٢٠ . نفذ البرنامج منتجا ملف بيانات يستخدم فى المشكلة التالية :

(٥٠) عدل برنامج فواتير العملاء المعطى فى مثال ١١ - ٢٠ بإضافة المعالم التالية :

١ - حدد أن كل سجل يجدد بإدخال رقم العميل من لوحة مفاتيح ، بدلا من محاولة تجديد السجلات التى لها موازنة حالية قائمة ( انظر تخطيط البرنامج الموضح فى مثال ١١ - ١٩ ) .

ب - أدخل جزءا لزيادة الموازنة القائمة ( أى إضافة دين جديد ) أثناء إعداد الفواتير لهذه الفترة . حدد مرة أخرى السجلات التى ستجدد بهذه الطريقة عن طريق إدخال أرقام العملاء المناسبة من لوحة المفاتيح .

استخدم البرنامج فى تشغيل ملف البيانات الناتج من آخر برنامج ، مع كتابة الديون الجديدة التالية :

Customer	New charge
Alan B. Adams	245.00
William D. McDonald	88.50

(٥١) عدل برنامج عمل الشفرة وفكها الموجود فى مثال ١١ - ٢٤ بحيث يمكن إدخال مفتاح key متعدد الأرقام مع استخدام أرقام متتالية لكل سطر متتالي . فمثلا اذا ما أدخل مفتاح ذو ثلاثة أرقام ، فيستخدم أول رقم لعمل الشفرة وفكها لأول سطر . والرقم الثانى للسطر الثانى ، والرقم الثالث للسطر الثالث . فإذا ما كان هناك أسطر أكثر من الأرقام الموجودة على المفتاح ، فتستخدم نفس هذه الأرقام مرة أخرى بنفس هذا الترتيب ، أى يستخدم الرقم الأول للسطر الرابع ، والرقم الثانى للسطر الخامس ، والرقم الثالث للسطر السادس .

اختبر البرنامج مستخدما العديد من الأسطر التى تختارها بنفسك .

(٥٢) عدل من محاكاة المباراة الموجودة فى مثال ( ٨ - ١٨ ) ، بحيث تحاكي عدداً محدداً من المباريات وتحفظ ( وتخزن ) ناتج كل مباراة فى ملف نصوص . وفى نهاية المحاكاه اقرأ ملف النصوص لتحديد نسبة الكسب والخسارة للاعب .

اختبر البرنامج بمحاكاة ١٠٠ مباراة متتالية . استخدم النتائج فى تقدير نسبة المكسب .

(٥٣) عدل برنامج إنتاج pig latin الموجود فى مثال ( ٩ - ٢٧ ) ، بحيث يمكن إدخال أسطر عديدة من النص عن طريق لوحة المفاتيح . خزن محتوى النص الإنجليزى فى ملف نصوص ، وخزن المحتوى المناظر له - والمكتوب بـ pig latin - فى ملف نصوص آخر .

أدخل فى البرنامج جزءا لإنتاج قائمة تسمح للمستفيد باختيار أى سمة من السمات التالية :

١ - إدخال نص جديد ، وتحويله إلى pig latin وتخزينه ( يتم تخزين كل من الملف الاصلى ولف pig latin ) .

ب - قراءة النص الذى سبق إدخاله من ملف النصوص وعرضه .

ج - قراءة pig latin المناظر للنص الذي سبق إدخاله وعرضه .

د - إنهاء الحسابات .

اختبر البرنامج ، مستخدماً أسطر اختيارية من أحد النصوص .

(٥٤) عدل نظام مراقبة المخزون المذكور في مثال (١٠ - ٢٨) ، بحيث إنه يشمل ملف بيانات يحتوى على سجلات فردية . أدخل جزءاً في البرنامج لأداء أى عملية من العمليات التالية :

أ - إضافة سجل جديد .

ب - تعديل سجل موجود ( بما في ذلك تعديلات في المعلومات الوصفية ، أو في ضبط كميات المخزون ) .

ج - حذف سجل .

د - إنتاج قائمة كاملة لكل العناصر الموجودة في المخزون .

هـ - إنهاء الحسابات .

اسمح باختيار العمليات الفردية من قائمة . اختبر البرنامج مستخدماً بيانات العينة الموجودة في مثال (١٠ - ٢٨) .

(٥٥) اكتب برنامجاً كاملاً بلغة البسكال ينتج ملف بيانات يحتوى على بيانات امتحانات الطلبة الموجودة في المشكلة رقم (٥٢ من الفصل التاسع) . اجعل كل مكون من مكونات الملف يكون سجلاً يحتوى على الاسم ودرجات الامتحان لكل طالب . نفذ البرنامج منتجاً ملف بيانات لاستخدامه في المشكلة التالية .

(٥٦) اكتب برنامجاً بلغة البسكال موجهاً للملفات ، يقوم بتشغيل درجات امتحانات الطلبة المعطاة في المشكلة رقم (٥٢ من الفصل التاسع) . اقرأ البيانات من ملف البيانات الذي أعد في المشكلة السابقة . بعد ذلك أنتج تقريراً يحتوى على الاسم ودرجات الامتحان ومتوسط الدرجة لكل طالب .

(٥٧) وسع البرنامج المكتوب في المشكلة السابقة لتحديد متوسط شامل لدرجة الفصل ، يتبعه انحراف متوسط كل طالب عن متوسط الفصل . اكتب المخرجات في ملف بيانات جديد . بعد ذلك أعرض المخرجات على هيئة تقرير به عناوين جيدة للتوضيح .

(٥٨) اكتب برنامجاً متداخلاً موجهاً للملفات يحفظ قائمة بأسماء وعناوين وأرقام الهاتف مرتبة ترتيباً أبجدياً ( طبقاً لإسم العائلة ) . ضع المعلومات المصاحبة لكل اسم في سجل منفصل . يحتوى البرنامج على قائمة تسمح للمستفيد بأن يختار أى عملية من العمليات التالية :

أ - إضافة سجل جديد .

ب - حذف سجل .

ج - تعديل سجل موجود .

د - استرجاع وعرض محتوى سجل يتناظر أسماً معيناً .

هـ - إنتاج قائمة كاملة بكل الأسماء والعناوين وأرقام التليفونات .

و - إنهاء الحسابات .

تأكد من إعادة ترتيب السجلات عند إضافة سجل ، أو عند حذف سجل بحيث تظل السجلات مرتبة ترتيباً أبجدياً بصفة دائمة .

(٥٩) اكتب برنامجاً ينتج ملف بيانات يحتوى على قائمة بالبلاد وعواصمها ، والموجود فى المشكلة ( ٥٦ من الفصل التاسع ) . ضع اسم كل بلد وعاصمتها فى سجل منفصل . نفذ البرنامج منتجاً ملف بيانات لاستخدامه فى المشكلة التالية .

(٦٠) اكتب برنامجاً متداخلاً يعمل بواسطة القائمة يتصل بملف البيانات الذى ينتج من المشكلة السابقة ، ويسمح بإداء إحدى العمليات التالية :

أ - تحديد عاصمة بلد محدد .

ب - تحديد البلد المعروف بعاصمته .

ج - إنهاء الحسابات .

(٦١) وسع البرنامج المكتوب فى المشكلة السابقة ليشمل المعالم الإضافية التالية :

أ - إضافة سجل جديد .

ب - حذف سجل .

تأكد أن قائمة البلاد تحفظ بالترتيب الأبجدي عند إضافة أو حذف أى سجل .

(٦٢) اكتب برنامجاً كاملاً بلغة البسكال يمكن استخدامه كمنقح بسيط موجه للأسطر . يجب أن يكون لهذا البرنامج المقدرات التالية :

أ - إدخال العديد من الأسطر من أحد النصوص ، وتخزينها فى ملف نصوص .

ب - سرد ملف النصوص .

ج - استرجاع وعرض سطر معين يتم تحديده عن طريق رقم السطر .

د - إدخال n سطراً .

هـ - حذف n سطراً .

و - حفظ النص الجديد المتقح ، وإنهاء الحسابات .

يجب أن يؤدى كل نشاط من هذه الأنشطة ، كاستجابة لأمر عبارة عن إدخال حرف واحد ، تسبقه علامة دولار . ويجب أن يتبع أمر الاسترجاع رقم صحيح بدون إشارة ليحدد السطر المطلوب استرجاعه . كما يمكن أن يلى أمر الحذف وأمر الإضافة رقم صحيح ، بدون إشارة إختياريا ، إذا ما كان مطلوباً حذف أو إضافة عدة أسطر متتالية .

ويجب أن يظهر كل أمر من هذه الأوامر فى سطر مستقل به ، وذلك ليقدّم وسيلة لتمييز الأوامر عن أسطر النص . ( يبدأ سطر الأمر بعلامة دولار ، يتبعها أمر مكون من حرف فردى ، ثم يلى ذلك رقم صحيح بدون إشارة (اختيارى) ومحدد لنهاية السطر ) .

ونوصى باستخدام الأوامر التالية :

- \$E إدخال نص جديد
- \$L سرد محتويات النص
- \$Fk استرجاع السطر رقم k .
- \$In إدخال n سطرًا بعد السطر رقم k .
- \$Dn حذف n سطرًا بعد السطر رقم k .
- \$S حفظ النص المنقح ، وإنهاء الحسابات .

(٦٣) وسع البرنامج المذكور فى المشكلة ( ٤٢ من الفصل العاشر ) ، بحيث تحفظ معلومات الفرق فى ملف بيانات ، بدلا من حفظها فى منظومة . ويجب أن يصبح كل مكون من مكونات الملف سجلا محتويا على بيانات إحدى الفرق . اجعل البرنامج محتويا على أجزاء تؤدى ما يلى :

أ - إدخال سجلات جديدة ( إضافة فرق جديدة ) .

ب - تجديد سجلات موجودة .

ج - حذف سجلات ( إلغاء فرق ) .

د - إنتاج تقرير تلخيصى لكل الفرق .



## الفصل الثاني عشر

### الفئات

## Sets

تعرف الفئة set رسمياً في البسكال بأنها مجموعة من عناصر البيانات البسيطة المرتبة ، والتي لها نفس النوع . وعلى هذا ... يمكن أن تكون الفئة عبارة عن مجموعة من عناصر البيانات الصحيحة ، أو عناصر البيانات الحرفية ، أو عناصر البيانات المتعددة .

ولكى يستخدم مفهوم الفئة ، يجب أن نعرف أولاً نوع الفئة . ويمكننا عند ذلك توضيح متغيرات من نوع الفئة تكون قيمتها الفردية عبارة عن عناصر من هذا النوع من أنواع الفئات . وفي واقع الأمر ... يمكن لمتغير واحد من نوع الفئة أن يمثل أى عدد من عناصر الفئة بما فيها العنصر الفارغ . وتقدم هذه الإمكانية طريقة بسيطة لنا لتحديد إذا ما وقع حدث أو كينونة entity في إحدى الفئات أو أكثر من فئة سبق تعريفها .

### 1. DEFINING A SET TYPE

#### ١ - تعريف نوع الفئة

نبدأ بمصاحبة مجموعة من عناصر البيانات البسيطة النوع والمرتبة ، مستخدمين تعريف TYPE كما سبق أن فعلنا ذلك . وهذا النوع من البيانات يعرف بأنه النوع الأساسي base type . وعلى هذا ... يمكننا تحديد النوع الأساسي بأنه :

`TYPE base type = (data item 1, data item 2, . . . ,data item n)`

أو:

`TYPE base type = 'first data item .. last data item`

ونوع الفئة الذى نريد تعريفه ، مقدم على ذلك بالنسبة للنوع الأساسي ، أى أن :

`set type = SET OF base type`

وعلى هذا ... فإن نوع الفئة يشير إلى نفس مجموعة عناصر البيانات مثل نوع الأساس .

وبعد تعريف نوع الفئة ، فإننا نوضح متغير نوع الفئة بالطريقة التالية :

`VAR set name : set type`

أو ، إذا كان مطلوباً متغيرات متعددة مختلفة من نوع الفئة ، فتوضح كمايلي :

`VAR set name 1, set name 2, . . . ,set name n : set type`

يمكن أن تمثل هذه المتغيرات من نوع الفئة بعض الفئات الجزئية للعناصر الموجودة في النوع الأساسي .

### مثال (١٢-١)

اعتبر التوضيح التالي :

```
TYPE sizes = (small,medium,large);
  shirtsizes = SET OF sizes;
VAR shortsleeve,longsleeve : shirtsizes;
```

في هذا المثال sizes هي نوع أساسي يحتوى على عناصر بيانات متعددة long و medium و small . نوع الفئة هو shirtsizes . ( لاحظ أن shirtsizes معرف بالنسبة للنوع الأساسي sizes ) . أخيرا فإن , longsleeve و shortsleeve متغيران من نوع الفئة لهما نفس النوع shirtsizes .

ويمكننا إذا رغبتنا أن نعرف أنواع فئات مختلفة عديدة من نفس النوع الأساسي . وعلى أية حال ... معظم التطبيقات البسيطة لا تحتاج إلى مثل هذا التعقيد .

### مثال (١٢-٢)

فيما يلي صيغة مختلفة من تعريفات نوع الفئة المعطاة في مثال (١٢ - ١) .

```
TYPE sizes = (small,medium,large);
  shirtsizes,dresssizes = SET OF sizes;
VAR shortsleeve,longsleeve : shirtsizes;
  shorthem,longhem : dresssizes;
```

لاحظ أن كلا من shirtsizes , dresssizes نوعى بيانات من نفس النوع الأساسي sizes . لاحظ أيضا أن longsleeve , shortsleeve متغيران من نوع الفئة ، وأن shorthem , longhem متغيران من نوع الفئة ، لهما نفس النوع مثل dresssizes .

يمكننا أيضا أن نعرف فئة بالنسبة لنوع بسيط ومرتب وقياسى ( وهو النوع الصحيح والنوع الحرفى ) أو نوع المدى الجزئى لنوع بسيط ومرتب وقياسى . فى مثل هذه الحالات يصبح نوع البيانات القياسى أو المدى الجزئى منه نوعا أساسيا .

### مثال (١٢-٣)

كل نوع من أنواع الفئات التالية معرف بالنسبة لنوع بسيط ، ومرتب ، وقياسى ، أو بالنسبة للمدى الجزئى المناظر .

```
TYPE numbers = SET OF integer;

TYPE digits = SET OF 0..9;

TYPE numchars = SET OF '0'..'9';

TYPE lowercase = SET OF 'a'..'b';
```

لاحظ أن ثانى نوع فئة يعرف فئات من النوع الصحيح ، بينما النوع الثالث يعرف فئات من النوع الحرفى .



## 2. CONSTRUCTING A SET

## ٢ - عمل الفئة

دعنا نوجه انتباهنا الآن إلى عمل فئات فردية . يمكن أن تحدد مثل هذه الفئات لمتغيرات من نوع الفئة من خلال لها . كما يمكن استخدامها أيضا كمؤشرات في بعض أنواع تعبيرات بوليان ( سيذكر المزيد عن ذلك فيما بعد ) .

يمكن أن تحتوى الفئة على أى عدد من العناصر من الفئة الأساسية المصاحبة . وتعد الفئة بكتابة العناصر الفردية على التوالي محصورة بين قوسين مربعين ، ومفصولة عن بعضها بواسطة فواصل . وعلى هذا ... تبدو الفئة الفردية على النحو التالي :

[set element 1, set element 2, . . . , set element n]

العناصر المحتواة تعرف بأنها أعضاء members في الفئة .

ويمكن أن تحتوى الفئة بالطبع على عنصر واحد ، كما أنه من الممكن أيضا إعداد فئة لا تحتوى على أى عنصر على الإطلاق . وهذه تعرف بأنها فئة صفرية null أو فئة فارغة empty . وتكتب على النحو التالي : [ ]

مثال (١٢-٤)

فيما يلي عدة فئات يمكن إعدادها من النوع الأساسي sizes المعروف في مثال (١٢-١)

```
[small,medium,large]
[medium,large]
[large,small]
[medium]
[]
```

لاحظ أن أعضاء الفئة لا تحتاج إلى أن تكون مرتبة ، كما هو موضح بالفئة الثالثة . لاحظ أيضا أن آخر فئة هي فئة خالية .

بعض أعضاء الفئة يمكن تمثيلها بواسطة متغيرات ، على أن تمثل هذه المتغيرات عناصر من نوع مناسب للنوع الأساسي . والأكثر من ذلك ... إذا ما كان بعض أعضاء الفئة عناصر متتالية ، فيمكن تمثيلها كمدى جزئى على النحو التالي :

first consecutive element .. last consecutive element

مثال (١٢-٥)

فيما يلي فئات إضافية أعدت من النوع الأساسي sizes ، وعدة متغيرات مناظرة لها .

```
TYPE sizes = (small,medium,large);
  shirtsizes = SET OF sizes;
VAR shirt,blouse : sizes;

[small..large]

[shirt]

[shirt,blouse]

[medium,large,blouse]
```

( لاحظ أن المتغيرات المستخدمة في هذا المثال هي متغيرات من النوع البسيط . وهي ليست متغيرات من نوع الفئة ، كما سبق ذكره في القسم السابق ) .

وعنصر الفئة لا يوجد في نفس الفئة إلا مرة واحدة فقط ، إلا أنه - على أية حال - من الممكن أن يحدد عنصر واحد مرتين أو أكثر إذا ما احتوى وصف الفئة على عناصر فئة مريحة ومتغيرات . ( يمكن أن يمثل المتغير عنصر فئة تم تحديده بالفعل ينتج عن ذلك ازدواج غير مرغوب فيه ) . في مثل هذه الحالات يهمل الوصف الزائد .

## مثال (١٢-٦)

اعتبر الفئة التالية :

[medium,large,blouse]

والموجودة في آخر مثال . تذكر أن large , medium هما عنصران من النوع الأساسي sizes . أما blouse فهو متغير variable من نفس نوع sizes . إذا ما تحدد أى من medium أو large للمتغير blouse ، فسوف يتكرر هذا العنصر داخل الفئة . وتفسر على ذلك الفئة بأن لها عضوين اثنين فقط هما large , medium . فإذا ما مثلت blouse العضو small ، فإن الفئة يصبح بها ثلاثة أعضاء ، وهي large , medium , small .

ويمكن ظهور مشاكل شبيهة إذا ما عبر عن بعض عناصر الفئة بأنها من نوع المدى الجزئي . فإذا كان العنصر الأول والعنصر الأخير من المدى الجزئي هما نفس العنصر ، فيفسر المدى الجزئي بأنه عنصر واحد . وأكثر من ذلك ... إذا ما اختلف العنصر الأول عن العنصر الأخير ... ولكن بالترتيب الخاطئ ( أى إذا ما جاء العنصر الأول بعد العنصر الأخير ) ، تعتبر الفئة فارغة .

## مثال (١٢-٧)

اعتبر الآن الفئة التالية :

[shirt .. blouse]

حيث blouse , shirt متغيران من نفس نوع sizes كما سبق توضيحهما في مثال (٥ - ١٢) . إذا ما مثل blouse , shirt نفس عنصر النوع الأساسي ( أى medium ) ، فإن الفئة تحتوى على عضو واحد فقط ، وهو العضو medium . كذلك إذا ما مثل shirt عنصر يأتى بعد blouse في النوع الأساسي ( أى إذا ما كان shirt يمثل large ، وكان blouse يمثل small ) ، فتعتبر الفئة خالية .

ويعد إعداد الفئة يمكن أن تحدد لمتغير من نوع الفئة . ويمكن تحقيق ذلك بالطريقة المعتادة بكتابة مايلي :

variable name := [set element 1, set element 2, . . . , set element n]

يجب أن يكون مفهوما أن الفئة التي تظهر في الناحية اليمنى يشار إليها بأنها عنصر بيانات أحادي القيمة .

ويجب أن يكون عنصر البيانات هذا من نفس نوع الفئة المتغير الذي يحدد له .

التخطيط الهيكلي التالي يوضح تحديد فئة داخل برنامج بسكال .

```
PROGRAM sample(input,output);
TYPE sizes = (small,medium,large);
   shirtsizes = SET OF sizes;
VAR shortsleeve,longsleeve : shirtsizes;
BEGIN
.
.
   shortsleeve := [small,large];
.
.
   longsleeve := [small,medium,large];
.
.
END.
```

### 3. OPERATIONS WITH SETS

### ٢ - العمليات التي تجرى على الفئات

هناك ثلاث عمليات مختلفة يمكن إجراؤها على الفئات ، وينتج عنها وجود فئة جديدة ونشير الى ناتج هذه العمليات ( أى الفئات الجديدة ) بأنها اتحاد union ، أو تقاطع intersection ، أو فرق الفئة set difference ، وذلك للفئتين الأصليتين على التوالي .

وتتطلب كل عملية عاملين ( أى فئتين ) من نفس النوع . والنتيجة على هذا تكون من نفس نوع العاملين .

واتحاد فئتين هو فئة جديدة تحتوى على كل أعضاء الفئتين الأصليتين ، ويستخدم المؤثر + لتحديد هذه العملية كما هو موضح أدناه .

### مثال (١٢-٩)

يوضح هذا المثال اتحاد فئتين .

```
PROGRAM sample(input,output);
TYPE sizes = (small,medium,large);
   shirtsizes = SET OF sizes;
VAR shortsleeve,longsleeve : shirtsizes;
BEGIN
.
.
   shortsleeve := [small] + [large];
.
.
   longsleeve := [small,medium] + [small,large];
.
.
END.
```

تنسب أول عبارة تحديد فى اتحاد الفئتين [ small ] , [ large ] وتحديدها لتغير من نوع السلسلة shortsleeve . وعلى هذا ... فإن shortsleeve يمثل الفئة ( [ small , large ] ) .

وبالمثل تتسبب عبارة التحديد الثانية في أن المتغير من نوع الفئة long sleeve يمثل الفئة [ small , medium , large ] )

وتقاطع فئتين عبارة عن فئة ، كل أعضائها تكون أعضاء مشتركة في كل من الفئتين الأصليتين . ويستخدم المؤثر \* في تحديد هذه العملية .

### مثال (١٠-١٢)

فيما يلي تقاطع لفئتين :

```
PROGRAM sample(input,output),
TYPE sizes = (small,medium,large);
  shirtsizes = SET OF sizes;
VAR shortsleeve,longsleeve : shirtsizes;
BEGIN
.
.
  shortsleeve := [small,medium] * [medium,large];
.
.
  longsleeve := [small] * [medium,large];
.
.
END.
```

تتسبب أول عبارة تحديد في تقاطع الفئتين ([ medium , large ] ) ([ small , medium ] ) ، وتحديد هذا التقاطع للمتغير من نوع الفئة shortsleeve . وعلى هذا ... فإن shortsleeve يمثل الفئة [ medium ] .

وبالمثل تتسبب عبارة التحديد الثانية في أن يمثل المتغير من نوع الفئة long sleeve الفئة الخالية [ ] ، حيث إن العاملين لا يحتويان على أعضاء مشتركين .

والفرق بين فئتين ، هو فئة توجد أعضاؤها في الفئة الأولى ولا توجد في الفئة الثانية . ويرمز لهذه العملية بالمؤثر - كما يتضح من المثال التالي :

### مثال (١١-١٢)

هذا المثال يوضح الفرق بين فئتين .

```
PROGRAM sample(input,output);
TYPE sizes = (small,medium,large);
  shirtsizes = SET OF sizes;
VAR shortsleeve,longsleeve : shirtsizes;
BEGIN
.
.
  shortsleeve := [small,medium] - [small,large];
.
.
  longsleeve := [small,medium,large] - [medium];
.
.
END.
```

تتسبب أول عبارة تحديد في حساب الفرق بين الفئتين ( [ small , medium ] ), ( [ small , large ] ) وتحديدًا لمتغير من نوع الفئة shortsleeve . وعلى هذا ... يمثل shortsleeve الفئة [ medium ] .

وبالمثل تتسبب عبارة التحديد الثانية في تحديد الفئة ( [ small , large ] ) لمتغير من نوع الفئة ، وهو longsleeve .

وعادة ما تدمج هذه العمليات مع عبارات تحديد الفئات لتعديل قيم متغيرات من نوع الفئة . وهذا صحيح بصفة خاصة مع عمليات الاتحاد والفرق .

#### مثال (١٢ - ١٢)

فيما يلي عدة عبارات تحديد تحتوى على استخدام لعمليات الفئات . يفترض أن المتغيرات ، longsleeve ، shortsleeve من نوع الفئة كما سبق تعريفهما في الأمثلة السابقة .

```
shortsleeve := shortsleeve + [medium];
longsleeve := longsleeve - [small];
shortsleeve := longsleeve + [large];
shortsleeve := longsleeve * [small,medium];
```

تتسبب العبارة الأولى في إضافة العنصر medium إلى الفئة الممثلة بواسطة متغير الفئة shortsleeve . ( فإذا كان medium عضوا في الفئة ، فلن يكون لهذه العبارة أى تأثير على الفئة ) .

والغرض من العبارة الثانية هو حذف العنصر small من الفئة التى يمثلها المتغير longsleeve ( إلا إذا لم يكن small موجودا من الأصل . وفى هذه الحالة لا يكون لهذه العبارة أى تأثير ) .

وفى العبارة الثالثة يضاف large إلى الفئة التى يمثلها ، وتحدد الفئة الجديدة للمتغير shortsleeve ( لاحظ أن longsleeve لم يتغير ) .

وأخيرا تتسبب آخر عبارة في تحديد الأعضاء المشتركين في كل من الفئة التى يمثلها longsleeve والفئة [ small , medium ] للمتغير shortsleeve .

#### 4. SET COMPARISONS

#### ٤ - مقارنة الفئات :

يمكن استخدام أربعة مؤثرات من السبعة مؤثرات العلاقية مع الفئات ، وذلك لتكوين تعبيرات من نوع بوليان . وهذه المؤثرات الأربعة وتفسيراتها عند استخدامها مع الفئات هي كما يلي :

المؤثر	تفسيره
=	تساوى الفئة ( أى أن كل فئتين تحتويان على نفس الأعضاء ، وينفس الترتيب ) .
<>	عدم تساوى الفئة ( لا تحتوى الفئتان على نفس الأعضاء بالضبط ) .

$\leq$  احتواء الفئة ( كل عضو في الفئة الأولى موجود داخل الفئة الثانية ) .

$\geq$  احتواء الفئة ( كل عضو في الفئة الثانية موجود في الفئة الأولى ) .

عند استخدام أى مؤثر من هذه المؤثرات ، فيجب أن تكون الفئتان من نفس النوع .

مثال (١٢-١٣)

فيما يلي عدة تعبيرات بوليان تحتوى على فئات . ( افرض أن أعضاء الفئة هي عناصر من نوع البيانات المتعدد ) .

`sizes = (small,medium,large)`

والتي سنطبق تقديمها في مثال (١٢ - ١٤)

القيمة	التعبير
خطأ	<code>[small,large] = [small,medium,large]</code>
صحيح	<code>[small,large] = [large,small]</code>
صحيح	<code>[small,medium,large] = [small..large]</code>
صحيح	<code>[small,medium] &lt;&gt; [medium]</code>
صحيح	<code>[small] &lt;= [small..large]</code>
خطأ	<code>[small,medium] &lt;= [small,large]</code>
خطأ	<code>[small..large] &lt;= [large]</code>
صحيح	<code>[ ] &lt;= [small..large]</code>
صحيح	<code>[small,medium,large] &gt;= [medium,large]</code>
صحيح	<code>[medium,large] &gt;= [medium,large]</code>
خطأ	<code>[medium] &gt;= [small,medium]</code>

لاحظ أن الفئة الصفرية محتواة داخل أى فئة أخرى . وعلى هذا ... فإن تعبيراً مثل التعبير التالي يكون دائماً

صحيحاً `true` . `[ ] <= [small,large]`

والمؤثرات العلاقية  $<$  ،  $>$  لها تفسيرات مختلفة بعض الشيء عند استخدامها مع المجموعات عن استخدامها

مع العناصر الأخرى . فإذا كانت كل من  $S1$  ،  $S2$  فئة ، فمن الممكن أن تكون التعبيرات  $S1 < S2$  ،  $S1 > S2$  خاطئة لبعض قيم عناصر هاتين الفئتين . وهذا لا يمكن حدوثه مع أنواع العناصر الأخرى .

مثال (١٢-١٤)

افرض أن لدينا فئتين ( مستقلتين ) `mutually exclusive` ، مثل `[small]` ، `[large]` . تعبيرات البوليان

التالية :

`[small] <= [large]`

`[small] >= [large]`

خاطئة `false` .

عند إجراء المقارنات بين الفئات ، يمكن التعبير عن عناصر الفئة كمتغيرات ، يحتوى نوعها الأساسى على أعضاء الفئة . أكثر من ذلك ... يمكن إجراء المقارنة بين فئة ومتغير من نوع الفئة له نفس نوع الأساس .

### مثال (١٢-١٥)

اعتبر التخطيط الهيكلى التالى :

```
PROGRAM sample(input,output);
TYPE letters = SET OF char;
VAR used,unused : letters;
    alpha : char;
    line : PACKED ARRAY [1..80] OF char;
BEGIN
.
.
FOR alpha := 'A' TO 'z' DO
    IF [alpha] <= used THEN write(alpha);
.
.
WHILE NOT ([line[1]] <= ['E','e']) DO
    BEGIN
        .
        .
        .
    END;
.
.
END.
```

فى أول عبارة إجرائية ( وهى عبارة FOR - TO ) تقارن الفئة التى تحتوى على قيمة alpha مع قيمة used . لاحظ أن alpha متغير من النوع الحرفى ، بحيث إن [alpha] تكون فئة محتوية على عنصر واحد من النوع الحرفى . كما أن used متغير من نوع الفئة أيضا من النوع الأساسى الحرفى . وعلى هذا ... فنحن نقارن فئة تحتوى على عناصر لها نفس نوع الأساسى ( أى أننا نقارن فئتين ، عناصر كل منهما حرفية ) .

لاحظ أن بوره FOR - TO تعتبر كل الحروف متراوحة من A الكبيرة ( قيمتها ٦٥ فى شفرة ASC II - انظر الملحق G ) إلى a الصغيرة ( قيمتها ١٢٢ فى شفرة ASC II ) . وهذا يعنى أن كل الحروف الأبجدية موجودة فى عملية المقارنة .

فى العبارة الإجرائية الثانية ( عبارة WHILE - DO ) تقارن الفئة التى تحتوى على قيمة line [1] مع الفئة [ 'E' , 'e' ] . تذكر أن line [1] هو عنصر من عناصر منظومة حرفية النوع ، وعلى ذلك فإن [ line [1] ] تكون فئة محتوية على عنصر فردى من النوع الحرفى ، وتحتوى الفئة التى تقارن معها [ 'E' , 'e' ] على عضوين من النوع الحرفى أيضا . وعلى هذا ... فإننا نقارن فئتين عناصرهما من نفس نوع الأساس .

وتقدم الأمثلة التالية برنامج بسكال كاملا ، مشتملا على استخدام الفئات

مثال (١٢-١٦)

تحليل أحد أسطر النص . افرض أننا نريد إدخال سطر من أسطر أحد النصوص داخل الكمبيوتر ، ثم نحدد أى الحروف الأبجدية موجودة داخله . يمكن تحقيق ذلك بسهولة باستخدام الفئات .

والطريقة هى قراءة السطر وتحليله عن طريق تحديد الحروف الموجودة فيه ، ثم كتابة كل الحروف التى نجدها فى السطر . نعتبر كل من الحروف الكبيرة والصغيرة ، وإن نأخذ فى الاعتبار أنواع الرموز الأخرى ( مثل التنقيط ) .

دعنا نقدم التوضيحات التالية :

```
TYPE letters = SET OF char;
VAR used,unused : letters;
    count : 0..80;
    alpha : char;
    line : PACKED ARRAY [1..80] OF char;
```

المتغيرات من نوع الفئة used , unused تمثل فئة الحروف المستخدمة والحروف غير المستخدمة على التوالى . كما يمثل line السطر الفعلى من النص .

بمعرفة هذه التوضيحات يمكن أن تظهر المجموعة الرئيسية للبرنامج على النحو التالى :

```
EGIN
  readinput; (* read in a line of text *)
  used := [];
  unused := ['A'..'Z','a'..'z'];
  FOR count := 1 TO 80 DO
    IF [line[count]] <= unused THEN
      BEGIN
        used := used + [line[count]];
        unused := unused - [line[count]]
      END;
  writeoutput (* write out the results of the analysis *)
END.
```

تشير العبارتان readinput , writeoutput إلى إجراءات تقرأ السطر وتكتب نتائج التحليل على التوالى :

وبعد إدخال السطر ، توضع قيم ابتدائية للمتغيرى الفئة used , unused ، وذلك بتحديد فئة صفرية للمتغير used ، وفئة تحتوى على كل الحروف الكبيرة والصغيرة للمتغير unused ، ثم يمكننا بعد ذلك فحص السطر حرفا حرفا . فإذا كان الحرف غير مستخدم ( أى إذا كانت الفئة [ line [count] ] محتواه داخل الفئة unused ) ، فيحدث تجديد لكل من used , unused بإضافة الحرف الحالى إلى used ، وحذفه من unused . ويحدث هذا التجديد عن طريق كتابة مايلى :

```
used := used + [line[count]];
unused := unused - [line[count]];
```



ثم تكتب بعد ذلك نتيجة هذا التحليل

وتعمل هذه الطريقة بصورة جيدة بالنسبة لسطر واحد من أسطر النص ، إلا أنه يجب إعادة بدء البرنامج في كل مرة ينتهي فيها السطر . دعنا على ذلك نعدل المجموعة الإجرائية الرئيسية ، بحيث يمكن تكرار تنفيذها حتى تدخل الكلمة " new " في بداية سطر جديد . وعلى هذا ... تعدل المجموعة الإجرائية الرئيسية لتأخذ الشكل التالي :

```
BEGIN
  readinput;  (* read in a line of text *)
  WHILE NOT (([line[1]] <= ['E','e']) AND
    ([line[2]] <= ['N','n']) AND
    ([line[3]] <= ['D','d'])) DO
    BEGIN
      used := [];
      unused := ['A'..'Z','a'..'z'];
      FOR count := 1 TO 80 DO
        IF [line[count]] <= unused THEN
          BEGIN
            used := used + {line[count]};
            unused := unused - {line[count]}
          END;
        writeoutput;  (* write out the results of the analysis *)
        readinput
      END
    END.
```

يسمح مكون WHILE - DO باستمرار الحسابات حتى يدخل سطر جديد يحتوى على الحرف c في بدايته ، والحرف n في العمود الثاني ، والحرف d في العمود الثالث . لاحظ أن استخدام مقارنة الفئات داخل هذا المكون تسمح بطريقة مريحة لاختيار كل من الحروف الكبيرة والحروف الصغيرة .

ويبدأ إجراء إدخال السطر readinput بوضع قيمة ابتدائية للمتغير line ، بحيث إنه يحتوى على فراغات فقط . بعد ذلك يقرأ النص الفعلي في الأماكن الفارغة . وتستمر القراءة حتى يكتشف محدد نهاية السطر . وفيما يلي هذا الإجراء كاملاً .

```
PROCEDURE readinput;
  (* this procedure reads in a line of text *)
  BEGIN
    FOR count := 1 TO 80 DO line[count] := ' ';
    writeln('Please enter a line of text below');
    count := 0;
    WHILE NOT eoln DO
      BEGIN
        count := count + 1;
        read(line[count])
      END;
    readln
  END;
```

إجراء المخرجات writeoutput يستخدم طريقة مختلفة . فهناك تجرى دورة خلال كل الحروف الهجائية ( الكبيرة

والصفيرة) وتكتب هذه الحروف كأعضاء في الفئة used . ويمكن كتابة الإجراء الكامل على النحو التالي :

```
PROCEDURE writeoutput;
(* this procedure writes out an analysis of a line of text *)
BEGIN
  writeln;
  write('Letters used:');
  FOR alpha := 'A' TO 'z' DO
    IF [alpha] <= used THEN write(' ',alpha);
  writeln;
  writeln
END;
```

تذكر أن alpha متغير من النوع الحرفي . وعلى هذا ... فكل من [ alpha ] , used يمثل فئة عناصرها من النوع الحرفي ) .

وفيما يلي برنامج البسكال كاملاً .

```
PROGRAM lettersused(input,output);

(* THIS PROGRAM READS A LINE OF TEXT AND
DETERMINES WHICH LETTERS ARE PRESENT *)

TYPE letters = SET OF char;
VAR used,unused : letters;
    count : 0..80;
    alpha : char;
    line : PACKED ARRAY [1..80] OF char;

PROCEDURE readinput;
(* this procedure reads in a line of text *)
BEGIN
  FOR count := 1 TO 80 DO line[count] := ' ';
  writeln('Please enter a line of text below');
  count := 0;
  WHILE NOT eoln DO
    BEGIN
      count := count + 1;
      read(line[count])
    END;
  readln
END;

PROCEDURE writeoutput;
(* this procedure writes out an analysis of a line of text *)
BEGIN
  writeln;
  write('Letters used:');
  FOR alpha := 'A' TO 'z' DO
    IF [alpha] <= used THEN write(' ',alpha);
```

(تكملة البرنامج في الصفحة التالية)

```

writeln;
writeln
END;

BEGIN (* main action block *)
  readinput;
  WHILE NOT (([line[1]] <= ['E','e']) AND
              ([line[2]] <= ['N','n']) AND
              ([line[3]] <= ['D','d'])) DO

    BEGIN
      used := [];
      unused := ['A'..'Z','a'..'z'];
      FOR count := 1 TO 80 DO
        IF [line[count]] <= unused THEN
          BEGIN
            used := used + [line[count]];
            unused := unused - [line[count]]
          END;
        writeoutput;
        readinput
      END
    END.

```

افترض الآن أن البرنامج يستخدم في تشغيل السطر التالي :

ينتج عن تنفيذ البرنامج الحواز التالي ( استجابات المستفيد موضوع تحتها خط ) .

```

Please enter a line of text below
Pascal is a structured programming language derived from ALGOL-60
-----
Letters used: A G L O P a c d e f g i l m n o p r s t u v

Please enter a line of text below
end

```

## 5. MEMBERSHIP TESTING

## ه - اختبار العضوية

تحتوى لغة البسكال على مؤثر علاقي إضافي أيضا IN يستخدم في عمل تعبيرات بوليان . ويمكن استخدام هذا المؤثر مع عناصر من نوع الفئة فقط . وهو مفيد بصفة خاصة ، حيث إنه يسمح لنا بتحديد ما إذا كانت القيمة محتواة داخل إحدى الفئات أم لا ( أى ما إذا كانت القيمة عضواً في الفئة أم لا ) .

ويجب استخدام المؤثر IN بالطريقة التالية لإنتاج تعبيرات بوليان .

*set element IN set*

يكون التعبير صحيحاً true إذا كان أول عنصر عضواً في العنصر الثاني ، وخاطئاً false إذا لم يتحقق ذلك . ويجب أن يناظر كل من العنصرين نفس النوع الأساسى . ويمكن أن يكون أول عنصر عنصر فئة فردياً أو متغيراً أو تعبيراً يمثل عنصر فئة . أما العنصر الثاني ، فهو فئة بصفة عامة ، أو متغيراً من نوع الفئة .

## مثال (١٢-١٧)

اعتبر الصيغة التالية لتوضيح الفئة الموجودة في مثال ( ١ - ١٢ )

```
TYPE sizes = (small,medium,large);
  shirtsizes = SET OF sizes;
VAR shortsleeve,longsleeve : shirtsizes;
  mysize : sizes
```

فيما يلي عدة تعبيرات بوليان توضيح استخدام المؤثر IN .

```
medium IN [small,medium,large]
medium IN [small,large]
mysize IN [small,medium]
mysize IN shortsleeve
```

التعبير الأول صحيح true ، والثاني خاطئ false ، إلا أن قيمة التعبير الثالث على أية حال تعتمد على القيمة التي تحدد للمتغير mysize . فإذا مامت mysize القيمة small أو medium ، فإن التعبير يكون صحيحا true ، وإلا فإنه يكون خاطئا false .

وبالمثل تعتمد قيمة آخر تعبير على القيم التي تحدد للمتغير mysize والمتغير shortsleeve . إذا مامت mysize عنصرا عضوا في الفئة التي يمثلها shortsleeve ، فإن التعبير يكون صحيحا true ، وإلا فإنه يصبح خاطئا false .

افترض على سبيل المثال أن mysize يمثل القيمة large . يكون التعبير صحيحا إذا مامت shortsleeve ، أي فئة من الفئات .

```
[small,medium,large]
[small,large]
[medium,large]
[large]
```

إلا أن التعبير يكون خاطئا لكل القيم الأخرى من shortsleeve .

وعادة ما يتكرر اختيار العضوية مع مكونات التحكم المختلفة داخل برنامج البسكال . وهذا يسمح لنا بأداء عمليات منطقية مختلفة طبقا لاختياراتنا ، وذلك عند تحقق شروط عضوية معينة من نوع الفئة فقط .

## مثال (١٢-١٨)

اعتبر التخطيط الهيكلي التالي :

( أنظر الصفحة التالية )

```

PROGRAM sample(input,output);
TYPE letters = SET OF char;
VAR vowels,consonants : letters;
    count,vowelcount,conscount : 0..80;
    line : PACKED ARRAY [1..80] OF char;

BEGIN
    vowels := ['A','E','I','O','U','a','e','i','o','u'];
    consonants := ['A'..'Z','a'..'z'] - vowels;

    vowelcount := 0;
    conscount := 0;

    FOR count := 1 TO 80 DO
        BEGIN
            IF line[count] IN vowels
                THEN vowelcount := vowelcount + 1;
            IF line[count] IN consonants
                THEN conscount := conscount + 1
        END;
    END;
END.

```

تحدد أول عبارتين قيما للمتغيرين من نوع الفئة vowels , consonants . وتختبر عبارتا IF ( الموجودتان داخل مكون FOR ) لمعرفة ما إذا كانت القيمة الحالية لعنصر منظومة من النوع الحرفي [ count ] line عضوا في الفئة التي يمثلها vowels أو التي يمثلها consonants .

ومن المفيد مقارنة هذا التخطيط مع التخطيط الهيكلي الموضح في مثال ( ١٢ - ١٥ ) . ففي المثال الحالي تختبر لمعرفة ما إذا ما كان عنصر بيانات قرئى عضوا في فئة أم لا . وفي الناحية الأخرى في المثال السابق ذكره نختبر ما إذا كانت فئة محتواه داخل فئة أخرى أم لا . ( لاحظ أن توافقية النوع الأساسي مطلوبة في كل من المثالين )

والمثال التالي يبين برنامج بسكال كاملاً يستخدم اختيار العضوية .

### مثال (١٢-١٩)

عدد الحروف المتحركة في السطر . دعنا نطور برنامجا بلغة البسكال ينفذ الأنشطة التالية :

- ١ - إدخال سطر من أحد النصوص داخل الكمبيوتر .
- ٢ - تحديد إجمالي عدد الحروف ( بما فيها الفراغات والتقطيع ) داخل السطر .
- ٣ - تحديد إجمالي عدد الحروف المتحركة ، وإجمالي عدد الحروف ، وعدد الحروف الساكنة داخل السطر .
- ٤ - كتابة إجمالي عدد الحروف ، وعدد الحروف المتحركة ، وعدد الحروف الساكنة .

اكتب البرنامج ، بحيث يكرر التنفيذ حتى يتم إدخال كلمة "end" في بداية سطر جديد ، مع السماح بإدخال end بحروف كبيرة أو حروف صغيرة .

نبدأ بتقديم التوضيحات التالية :

```
TYPE letters = SET OF char;
VAR vowels, consonants : letters;
    count, charcount, vowelcount, conscount : 0..80;
    line : PACKED ARRAY [1..80] OF char;
```

يمثل المتغيران من نوع الفئة الحروف المتحركة والحروف الساكنة فئتي الحروف المتحركة والحروف الساكنة على التوالي . وسوف توجد الحروف الصغيرة والكبيرة في كل فئة من الفئتين . وتمثل المتغيرات الصحيحة , conscount , charcount , vowelcount إجمالي عدد الرموز الموجودة في السطر ( بما في ذلك الفراغات والتنقيط ) وعدد الحروف المتحركة وعدد الحروف الساكنة على التوالي . وأخيرا فإن المنظومة الحرفية line تمثل السطر الفعلي من النص .

دعنا نعتبر الآن المجموعة الإجرائية الرئيسية ، إذا ما أدخلنا جزءا في البرنامج لتكرار التنفيذ ، فيمكن كتابة المجموعة الرئيسية كما يلي :

```
BEGIN
    vowels := ['A','E','I','O','U','a','e','i','o','u'];
    consonants := ['A'..'Z','a'..'z'] - vowels;
    readinput;
    WHILE NOT ((line[1] IN ['E','e']) AND
                (line[2] IN ['N','n']) AND
                (line[3] IN ['D','d'])) DO
        BEGIN
            vowelcount := 0;
            conscount := 0;
            FOR count := 1 TO 80 DO
                BEGIN
                    IF line[count] IN vowels
                        THEN vowelcount := vowelcount + 1;
                    IF line[count] IN consonants
                        THEN conscount := conscount + 1;
                END;
            writeoutput;
            readinput;
        END
    END.
```

تحدد أول عبارتين قيم المتغيرات من نوع الفئة vowel , consonants على التوالي . وتستخدم هذه القيم كنمطيات لاختبار العضوية . وعلى هذا ... فلن تتغير أثناء تنفيذ البرنامج .

ونرى عبارة readinput تلي عبارتي تحديد الفئتين . وتشير هذه العبارة إلى إجراء ، يتسبب في قراءة سطر من النص داخل الكمبيوتر ، وعد العدد الإجمالي للرموز .

اعتبر الآن مكون WHILE - DO ، وهو يقع في قلب البرنامج في الواقع . يبدأ المكون بتحديد قيمة ابتدائية صفر للعدادات vowelcount , conscount . وبعد ذلك يفحص البرنامج محتويات السطر رمزا رمزا ، فإذا ما كان أحد

الرموز للحروف المتحركة ( وبصورة أكثر تحديداً ... إذا ما كان الرمز عضواً في فئة الرموز المحددة للحروف المتحركة ) يزداد عدد vowel بمقدار ١ . وبالمثل إذا ما كان أحد الرموز للحروف الساكنة ، فإن عدد الحروف الساكنة يزداد بمقدار .

وبعد ذلك تكتب نتيجة التحليل عن طريق الاتصال بالإجراء writeoutput ، ثم يتم إدخال سطر جديد داخل الكمبيوتر عن طريق إجراء readinput . وتعاد هذه الدورة حتى تدخل كلمة end في بداية سطر جديد .

والإجراء المستخدم لقراءة سطر جديد من النص وتحديد إجمالي عدد الرموز ، يمكن أن يكتب على النحو التالي :

```
PROCEDURE readinput;
(* this procedure reads in a line of text *)
BEGIN
  writeln('Please enter a line of text below');
  count := 0;
  WHILE NOT eoln DO
    BEGIN
      count := count + 1;
      read(line[count]);
    END;
  readln;
  charcount := count
END;
```

وهذا الإجراء مباشر ، ولا يحتاج إلى أي مناقشة .

كما أننا إجراء المخرجات writeoutput مباشر أيضاً . وهذا الإجراء يكتب القيم النهائية للثلاثة عدادات con- charcount , vowelcount , scount . وعلى هذا ... يمكننا كتابة هذا الإجراء كما يلي :

```
PROCEDURE writeoutput;
(* this procedure writes out an analysis of a line of text *)
BEGIN
  writeln;
  writeln('Number of characters: ',charcount:2);
  writeln('Number of vowels : ',vowelcount:2);
  writeln('Number of consonants: ',conscount:2);
  writeln
END;
```

وفيما يلي البرنامج كاملاً :

```
PROGRAM vowelcount(input,output);
(* THIS PROGRAM COUNTS THE TOTAL NUMBER OF CHARACTERS,
   THE NUMBER OF VOWELS AND THE NUMBER OF CONSONANTS
   APPEARING IN A LINE OF TEXT *)
TYPE letters = SET OF char;
VAR vowels,consonants : letters;
    count,charcount,vowelcount,conscount : 0..80;
    line : PACKED ARRAY [1..80] OF char;
PROCEDURE readinput;
(* this procedure reads in a line of text *)
BEGIN
  writeln('Please enter a line of text below');
  count := 0;
  WHILE NOT eoln DO
    BEGIN
```

(تكملة البرنامج في الصفحة التالية)

```

        count := count + 1;
        read(line[count])
    END;
    readln;
    charcount := count
END;

PROCEDURE writeoutput;
(* this procedure writes out an analysis of a line of text *)
BEGIN
    writeln;
    writeln('Number of characters: ',charcount:2);
    writeln('Number of vowels   : ',vowelcount:2);
    writeln('Number of consonants: ',conscount:2);
    writeln
END;

BEGIN (* main action block *)
    vowels := ['A','E','I','O','U','a','e','i','o','u'];
    consonants := ['A'..'Z','a'..'z'] - vowels;
    readinput;
    WHILE NOT ((line[1] IN ['E','e']) AND
                (line[2] IN ['N','n']) AND
                (line[3] IN ['D','d'])) DO
        BEGIN
            vowelcount := 0;
            conscount := 0;
            FOR count := 1 TO 80 DO
                BEGIN
                    IF line[count] IN vowels
                        THEN vowelcount := vowelcount + 1;
                    IF line[count] IN consonants
                        THEN conscount := conscount + 1
                END;
            writeoutput;
            readinput
        END
    END.
END.

```

اعتبر الآن ما يحدث عندما ينفذ هذا البرنامج . افرض على سبيل المثال أننا أدخلنا السطرين التاليين :

```

Pascal is a structured programming language derived from ALGOL-60
end

```

ينتج عن تنفيذ البرنامج الحوار التالي (استجابات المستخدم موضوع تحتها خط) .

```

Please enter a line of text below
Pascal is a structured programming language derived from ALGOL-60
-----
Number of characters: 65
Number of vowels   : 20
Number of consonants: 34

Please enter a line of text below
end

```



## Review Questions

## أسئلة مراجعة :

- (١) ماهى الخواص الأساسية للفئة ؟ وكيف تختلف الفئة عن المنظومة ؟ وكيف تختلف الفئة عن السجل ؟
- (٢) ماذا يعنى النوع الأساسى ؟ وكيف يعرف النوع الأساسى ؟
- (٣) ماذا يعنى نوع الفئة ؟ وكيف يختلف نوع الفئة عن النوع الأساسى ؟ وكيف يعرف نوع الفئة ؟
- (٤) أى نوع من أنواع البيانات يمكن أن يوجد داخل الفئة ؟
- (٥) هل يمكن تعريف أنواع فئات مختلفة لها نفس النوع الأساسى ؟
- (٦) ماهو الغرض من المتغير من نوع الفئة ؟ وماهو نوع عنصر البيانات الذى يمثله هذا المتغير ؟
- (٧) لخص قواعد إعداد الفئة ؟
- (٨) ماهو أقل عدد من العناصر يجب أن يوجد فى الفئة ؟
- (٩) عند إعداد الفئة ، هل يجب أن تحدد العناصر الفردية للفئة بترتيب معين ؟
- (١٠) ماهى القيود الموضوعة على استخدام متغيرات من نوع الفئة عند إعداد الفئة ؟
- (١١) عند إعداد الفئة ، ماهى العناصر المتتالية فى الفئة التى يعبر عنها بالمدى الجزئى ؟
- (١٢) عند إعداد الفئة ، ماذا يحدث إذا ماتم تحديد نفس عنصر الفئة أكثر من مرة واحدة ؟
- (١٣) افترض أن مواصفات الفئة تحتوى على مدى جزئى واحد ، ومواصفات المدى الجزئى مكتوبة بترتيب خاطئ . كيف يفسر ذلك ؟ وضح إجابتك .
- (١٤) لخص قواعد تحديد فئة لمتغير من نوع الفئة . ماهى القيود الموضوعة بالنسبة لتوافقية النوع ؟
- (١٥) ماذا يعنى اتحاد فئتين ؟ كيف يمكن أداء هذه العملية فى البسكال ؟ وماهى القيود الموضوعة على العناصر التى تجرى عليها هذه العملية ؟
- (١٦) ماذا يعنى تقاطع فئتين ؟ كيف يمكن أداء هذه العملية فى البسكال ؟ وماهى القيود الموضوعة على العناصر التى تجرى عليها هذه العملية ؟
- (١٧) ماذا يعنى الفرق بين فئتين ؟ كيف يمكن أداء هذه العملية فى البسكال ؟ وماهى القيود الموضوعة على العناصر التى تجرى عليها هذه العملية ؟
- (١٨) ماهى المؤثرات العلاقية التى يمكن استخدامها مع عناصر من نوع الفئة ؟ وماهى المؤثرات العلاقية المستخدمة فى ذلك ؟ وكيف يفسر كل منها ؟

(٢٠) مانوع العناصر التي يمكن استخدامها مع المؤثر العلاقي IF ؟ قارن ذلك باستخدام المؤثرات العلاقية الأخرى مع عناصر من نوع الفئة .

### مسائل محلولة :

```

(a) VAR vowels, consonants : SET OF char;
(b) VAR caps : SET OF 'A'..'Z';
(c) TYPE capitals = 'A'..'Z';
    VAR caps : SET OF capitals;
(d) TYPE capitals = 'A'..'Z';
    uppers = SET OF capitals;
    VAR caps : uppers;
(e) VAR movement : SET OF (north,south,east,west);
(f) TYPE compass = (north,south,east,west);
    VAR movement : SET OF compass;
(g) TYPE compass = (north,south,east,west);
    direction = SET OF compass;
    VAR nextmove, lastmove : direction;

```

(a) [mon..fri]	(d) [fri..mon] + [wed]
(b) [mon..fri] + [wed]	(e) [tue] - [mon..sat]
(c) [mon..fri] - [wed]	(f) [sun..wed] * [tue..sat]
(a) [mon,tue,wed,thu,fri]	(d) [wed]
(b) [mon,tue,wed,thu,fri]	(e) [ ]
(c) [mon,tue,thu,fri]	(f) [tue,wed]

```
workdays := [mon..sat] - restdays;
```

```

(c) workdays := [ ];
    FOR today := mon TO thu DO
        workdays := workdays + [today];
    restdays := workdays * [thu..sat];

(a) workdays = {sun,mon,wed,thu,fri,sat}
(b) workdays = {mon,wed,fri,sat}
(c) workdays = {mon,tue,wed,thu}
    restdays = {thu}

```

(٢٤) حدد قيمة كل تعبير من تعبيرات بوليان التالية طبقا للتوضيحات التالية :

```

TYPE weekdays = (sun,mon,tue,wed,thu,fri,sat);
    days = SET OF weekdays;
VAR workdays, restdays : days;
    today : weekdays;

(a) [sun,sat] = [sun..sat]
(b) [tues,fri] >= [ ]
(c) [mon,wed,fri] <= [mon..fri]
(d) [fri,mon..wed] = [mon,wed,fri]
(e) workdays := [mon..fri];
    .....
    workdays <> [mon,tue,wed,thu,fri]
(f) today := fri;
    restdays := [fri,sat,sun];
    .....
    [today] <= restdays

(g) mon IN {mon,wed,fri}
(h) sun IN [sun..sat]
(i) sat IN [mon,wed,fri]
(j) today := wed;
    .....
    today IN [mon,wed,fri]
(k) today := sat;
    workdays := [mon..fri];
    .....
    today IN workdays

(a) false      (e) false      (i) false
(b) true       (f) true       (j) true
(c) true       (g) true       (k) false
(d) false      (h) true

```

## Supplementary Problems

## مشاكل متكاملة :

(٢٥) حدد العناصر الموجودة في كل فئة من الفئات التالية طبقا للتوضيحات التالية :

```

TYPE notes = (do,re,mi,fa,sol,la,ti);
    song = SET OF notes;

(a) [re..la]
(b) [do..fa] + [re..la]
(c) [do..fa] - [re..la]
(d) [do..fa] * [re..la]

(e) [do..fa] - [la..re]
(f) [re,fa,sol] - [re..la]
(g) [re,fa,sol] * [ ]
(h) [do,re,mi] + [re]

```

(٢٦) حدد الناتج من كل عبارة من عبارات التحديد التالية ، طبقاً للتوضيحات التالية :

```
TYPE notes = (do, re, mi, fa, sol, la, ti);
  song = SET OF notes;
VAR ballad, disco : song;
  tone : notes;
```

- (a) ballad := [do..fa] \* [re, fa];
- (b) disco := [do..fa] - [mi];
- (c) tone := ti;

.  
.  
.

ballad := [re..fa, la] + [tone];

- (d) disco := [do, mi, sol];

.  
.  
.

ballad := disco + [re, ti];

- (e) disco := [];
- FOR tone := la DOWNTON re DO
- disco := disco + [tone];

(٢٧) حدد قيمة كل تعبير من تعبيرات بولياني التالية ، طبقاً للتوضيحات التالية :

```
TYPE coins = (penny, nickel, dime, quarter, half, dollar);
  change = SET OF coins;
VAR money : coins;
  wine, whiskey, song : change;
```

- (a) [penny, nickel] <= [penny, quarter]
- (b) [nickel, dime, quarter] = [nickel..quarter]
- (c) [penny..dollar] = [dollar..penny]
- (d) [nickel..quarter] >= [dime, quarter]
- (e) [half..nickel] <> []
- (f) dollar IN [penny..dollar]
- (g) money := dollar;

. . . . .  
[money] <= [penny..dollar]

- (h) money := dollar;

. . . . .  
money IN [penny..dollar]

- (i) wine := [penny, nickel, dime];

. . . . .  
dollar IN wine

- (j) whiskey := [penny, nickel, dime];

. . . . .  
dime IN whiskey

- (k) wine := [penny, nickel, dime];

song := [dime..dollar];

. . . . .  
dime IN wine \* song

- (l) money := penny;

song := [dime..dollar];

. . . . .  
money IN song

(٢٨) فيما يلي العديد من العبارات أو مجموعات العبارات التي تحتوى على فئات أو أعضاء فئات ، طبقا للتوضيحات المعطاة فى المشكلة السابقة . بعض هذه العبارات غير صحيحة . عرف كل الأخطاء .

- ```
(a) money := dime;
    .
    .
    .
    WHILE money IN {penny,nickel,dime} DO
    BEGIN
    .
    .
    .
    END;
```
- ```
(b) money := dime;
    .
    .
    .
    WHILE money <= {penny,nickel,dime} DO
    BEGIN
    .
    .
    .
    END;
```
- ```
(c) song := [];
    .
    .
    .
    FOR money := penny TO dollar DO
    song := song + money;
```
- ```
(d) IF NOT (quarter IN wine) THEN
    wine := wine + [quarter];
```
- ```
(e) IF [penny] <= song THEN song := song - [penny];
```
- ```
(f) IF whiskey IN [penny..quarter] THEN
    whiskey := whiskey + [half];
```

## Programming Problems

## مشاكل برمجة :

(٢٩) عدل البرنامج الموجود فى مثال ( ١٢ - ١٦ ) بالطرق التالية :

أ - استخدام اختبار عضوية داخل البرنامج ( أى إحلال تعبيرات بوليان بتعابير مكافئة لاستخدام المؤثر ( IN ) .

ب - تحديد العناصر غير الحرفية ( أى الأرقام وعلامات التنقيط وخلافه ) . الموجودة بالإضافة الى الرموز الموجودة .

ج - كتابة القوائم الأربع التالية بعد تحليل كل سطر .

١ - الحروف الموجودة فى سطر معين من أسطر النص .

٢ - الرموز غير الحرفية الموجودة في سطر معين من أسطر النص .

٣ - الحروف غير الموجودة في النص .

٤ - الرموز غير الحروف غير الموجودة .

اختبر البرنامج مستخدماً عدة أسطر مدخلات من اختيارك الشخصي .

(٣٠) وسع البرنامج الموجود في مثال ( ١٢ - ١٩ ) ليحتوى على المعالم التالية :

١ - أحسب عدد عناصر كل الحروف المتحركة التى تظهر داخل السطر .

ب - تحديد عدد الكلمات الموجودة في السطر ( ملاحظة عدد الفراغات الموجودة في السطر )

ج - تحديد متوسط طول كل كلمة داخل السطر .

اختبر البرنامج ، مستخدماً العديد من الأسطر التى تختارها بنفسك

(٣١) أكتب برنامجاً متداخلاً بلغة البسكال ، يحول التاريخ الذى تم إدخاله في صورة yy - dd - mm (مثل - 22 - 4 69) إلى رقم صحيح يحدد عدد الأيام بعد أول يناير ١٩٦٩ ، مستخدماً الطريقة المذكورة في المشكلة رقم (٥٠) من الفصل السادس ( الجزء ٢ ) على أن يحتوى البرنامج على اختبارات للخطأ ، بحيث يمكن اكتشاف بيانات المدخلات الخاطئة ، وإنتاج رسالة خطأ مناسبة . استخدم تكوين بيانات من نوع الفئة في جزء اختبار الخطأ .

(٣٢) أعد كتابة البرنامج المتداخل الخاص بلعبة tic - tac - toe الموجود في المشكلة رقم ( ٥٠ من الفصل السابع ( الجزء ١ ) ، بحيث يشمل البرنامج جزءاً للتأكد من الخطأ الخاص ببيانات المدخلات . يجب أن يختبر هذا الجزء النقل الخاطئ والاستجابات الخاطئة للفتات المدخلات . استخدم تكوين بيانات من نوع الفئة في هذا الغرض .

(٣٣) أعد كتابة برنامج منتج pig latin المذكور في المشكلة رقم ( ٥٠ من الفصل التاسع ) ، بحيث يستخدم تكوين بيانات من نوع الفئة عند اختبارات علامات التثنية وأصوات الأحرف المزبوجة والحروف الكبيرة . هل استخدام الفئة هو أفضل طريقة لتنفيذ كل من هذه الاختبارات ؟

(٣٤) تصف المشكلة رقم ( ٥٢ من الفصل التاسع ) تطبيق برمجة يحسب فيه متوسط درجات الطلاب في امتحانات البسكال ، وتحديد متوسط درجة الفصل . افرض الآن أنه تم تعريف خمسة مجالات عديدة ، بحيث يقع المتوسط الشامل للفصل في منتصف المجال الثالث . ويمكن أن يحدد حرف كمعدل لكل طالب ، وذلك طبقاً للمدى الخاص الذى يقع فيه متوسط كل طالب ... وعلى هذا ... فإن أول مدى ( أعلى مدى ) يمثل A ، والثاني يمثل B وهكذا .

واحدى الطرق لعمل المجالات العددية الفردية هي كما يلي : دع CA تمثل المتوسط الشامل للفصل . واحسب خارج القسمة .

$$Q = \frac{(100 - CA)}{3}$$

ثم احسب المدى كما يلي :

first range (A):	$(CA + 2Q)$	to	100
second range (B):	$(CA + Q)$	to	$(CA + 2Q)$
third range (C):	$(CA - Q)$	to	$(CA + Q)$
fourth range (D):	$(CA - 2Q)$	to	$(CA - Q)$
fifth range (F):	below $(CA - 2Q)$		

وعلى هذا إذا كان المتوسط الشامل للفصل ٧٠ ٪ ، فإن  $Q$  تساوى  $10 = (100 - 70) / 3$  ، وتصبح المجالات كما يلي :

- A: 90 to 100
- B: 80 to 90
- C: 60 to 80
- D: 50 to 60
- F: below 50

اكتب برنامجا متداخلا بلغة البسكال ينفذ هذه العملية ، مفترضا أوزانا (ترجيما) متساوية لكل الامتحانات .  
أستخدم كل من تكوينات بيانات من نوع السجل ، ومن نوع الفئة فى البرنامج . اختبر البرنامج مستخدما البيانات المعطاه فى المشكلة رقم (٥٢ من الفصل التاسع) .

(٣٥) فيما يلي طريقة للحصول على قائمة بالأعداد الأولية التى تقع داخل المدى الذى يتراوح من 2 إلى  $n$  .

١ - إنتاج قائمة مرتبة بالأرقام الصحيحة من 2 إلى  $n$  .

ب - أداء العمليات التالية لبعض الأرقام الخاصة الموجود فى القائمة .

١ - كتابة الرقم وإضافته إلى قائمة الأعداد الأولية .

٢ - نقل كل الأرقام المتتالية التى تقبل القسمة على  $i$  .

ج - إعادة الجزء (b) لكل قيمة متتالية من قيم  $i$  ، بدءا بقيمة  $i = 2$  ، ومنتهيا بأخر رقم متيق .

عادة ما يشار إلى هذه الطريقة بأنها منخل اراتوستينز Sieve of Eratosthenes .

اكتب برنامجا بلغة البسكال يستخدم هذه الطريقة فى تحديد الأعداد الأولية الموجودة فى القائمة ، والتى تتراوح من 1 إلى  $n$  حيث  $n$  هى كمية يتم إدخالها للبرنامج .

استخدم تكوينات بيانات من نوع الفئة داخل البرنامج .

(٣٦) ترغب إحدى وكالات العمل فى حفظ قائمة باستخدام الكمبيوتر للوظائف المتاحة حاليا . وداخل هذه القائمة يتم وصف كل وظيفة براتبها الشهرى ، أو بأى صفات أخرى تميزها .

وعندما يتصل أحد الأشخاص الباحثين عن عمل بهذه الوكالة ، يتم إدخال معلومات وصفية عن هذا الشخص داخل الكمبيوتر لتقارن مع الخواص التى تميز الوظائف المتاحة ، ثم ينتج عند ذلك قائمة بالوظائف ذات الرواتب المرتفعة الجيدة ، والتى تناسب هذا الشخص .

وتستخدم الصفات التالية في تمييز مهارات العمل المطلوب .

Attribute	Job Skill
A	accounting
B	business
C	computer programming
D	dental technology
E	engineering/technical
F	food service
M	medical technology
P	personnel administration
R	receptionist
S	sales
T	typing/word processing
U	unskilled labor

كما يكون مطلوبا صفات أخرى تحدد المستوى التعليمي المطلوب ، وهي :

Attribute	Educational Level
1	high school
2	vocational
3	college
4	postgraduate

اكتب برنامجا بلغة البسكال يمكنه أن يلبى احتياجات وكالة العمل هذه . استخدم تكوينات بيانات من نوع الفئة لعمل التوافق المطلوب بين الشخص والوظائف .

اختبر البرنامج مستخدما البيانات التالية :

Positions Available			
Position Number	Monthly Salary, \$	Required Job Skills	Education Level
1	625	U	1
2	1350	A,T	3
3	900	S	3
4	2400	E,B	4
5	450	F	1
6	1100	P,T	1
7	1700	D,T,R	1
8	200	M	2

Potential Employees			
Client Number	Minimum Req'd Salary, \$	Job Skills	Education Level
1	800	R,T	1
2	1200	A,C,T	3
3	2000	E,B,C	4
4	400	U	1
5	800	S,P,T	3



(٣٧) عدل البرنامج المكتوب في المشكلة السابقة ، بحيث تخزن المعلومات التي تصف الوظائف المتاحة في ملف بيانات . استخدم تكوينات بيانات من نوع السجل لتمثيل كل وظيفة من الوظائف المتاحة . أدخل في البرنامج جزءا لإضافة سجلات ، وحذف سجلات ، والتغيير سجلات ، وإسرد كل السجلات .

(٣٨) ترغب إحدى الجامعات في إنتاج إجراء تسجيل للطلاب آليا ، حيث يستخدم كل قسم جهاز كمبيوتر صغيرا خاصا به ، وقاعدة بيانات خاصة به لهذا الغرض . تحتوي قاعدة البيانات على قائمة بكل المقررات التي يجب أن يدرسها الطالب ليحصل على درجته الجامعية ، وتحتوي على المتطلبات السابقة أيضا . وسوف تخزن هذه القائمة بالطبع في ملف بيانات .

ويجب أن تدرس المقررات بترتيب عددي ، فيما عدا أن الطلاب لا يستطيعون التسجيل في مقرر إلا إذا نجحوا في المقررات الأخرى التي تعتبر متطلبات أساسية لهذا المقرر . ويصبح هذا الموقف معقدا بعض الشيء ، نظرا للحقيقة التي تقول إن الطلبة يرسبون في مقرر ، ويخرجون من التسلسل على هذا . كما أن بعض الطلبة ينتقلون أيضا إلى برنامج دراسة ، وعلى هذا ... فلا يدرسون كل المتطلبات السابقة .

اكتب برنامجا بلغة البسكال يسمح لكل طالب في أحد الأقسام أن يسجل نفسه في ثلاثة مقررات جديدة ، كل فصل دراسي مختارا من المقررات الموجودة في ملف البيانات ( افترض أن كل المقررات الموجودة في ملف البيانات تعرض كل فصل دراسي )

يجب اختيار المقررات طبقا لتسلسلها العددي المناسب ، مع افتراض أن الطالب أتم كل المتطلبات السابقة للمقرر الذي يسجل له . استخدم تكوينات بيانات من نوع الفئة لأداء هذا الاختبار .

اختبر البرنامج مستخدما البيانات التالية :

Courses

101	201 (101)	301 (204)	401 (301)
102	202	302 (102,202)	402 (302)
103	203 (106)	303	403
104 (101)	204 (201)	304 (301)	404 (304)
105 (102)	205	305 (302)	405 (205)
106	206 (102,203)	306	406 (102,306)

Student Records

Name	Courses Taken
Smith	101,103
Brown	101,102,103,104,105,202
Richardson	101,102,103,104,105,106,201,203,204
Davis	101,102,104,105,201,203,204,206,301,304
Thomas	101,102,103,104,105,106,201,202,203,204,205,206,302,303,305,306,402,405



## الفصل الثالث عشر

### القوائم والمشيريات (١)

## Lists and Pointers

اهتمنا في الفصول من الفصل التاسع وحتى الفصل الثاني عشر بأنواع مختلفة متعددة لتكوينات البيانات ، وهي : المنظومة ، والسجل ، والملف ، والفئة .

وبالرغم من أن كل نوع من هذه الأنواع للبيانات له خواصه الخاصة به ، فكلها تشترك في خواص معينة . فمثلا أقصى عدد لمكونات منظومة أو سجل أو فئة يحدد في توضيح البيانات ، ويظل ثابتا طوال فترة تنفيذ البرنامج . أكثر من هذا ... ترتب المكونات الفردية داخل أى نوع من أنواع تكوينات البيانات في ترتيب ثابت بالنسبة لبعضها . ( وهذا صحيح بالنسبة لكل تكوينات البيانات ، بما فيها الملفات ) . وتميل هذه الخواص إلى تقييد المنفعة من أنواع تكوينات البيانات في بعض التطبيقات .

وبصفة خاصة تتطلب بعض التطبيقات استخدام قوائم ، تكون مكوناتها متصلة مع بعضها بواسطة مشيرات pointers . وعلى هذا ... يشار إلى مثل هذه القوائم بأنها قوائم متصلة linked lists . وتستخدم القوائم المتصلة على سبيل المثال مع المترجمات ، ونظم التشغيل ، ونظم إدارة قواعد البيانات . وعلى هذا ... فإننا نعيد انتباهنا الآن إلى استخدام القوائم المتصلة في البسكال .

### 1. PRELIMINARIES

#### ١ - أساسيات

الفكرة الأساسية للقائمة المتصلة هي أن كل عنصر فردي داخل القائمة يحتوى على مشير يحدد المكان الذى يمكن أن يوجد فيه العنصر التالى له . وعلى هذا ... فإن الترتيب النسبى للمكونات يمكن تغييره ببساطة . وذلك بتغيير المشير ، بالإضافة إلى ذلك ... يمكن إضافة المكونات الفردية بسهولة إلى القائمة ، أو حذفها منها ، وذلك بتغيير المشير أيضا . وعلى هذا ... فإن القائمة المتصلة لا تقتصر على حد أقصى لعدد مكوناتها . ومثل هذه القائمة يمكن توسيعها أو تضيقها بالنسبة لحجمها . والبرنامج الذى يستخدمها .

مثال (١٣-١)

يوضح الشكل (a) ١٣-١ قائمة متصلة تحتوى على ثلاثة مكونات . كل مكون من هذه المكونات الثلاثة يحتوى على عنصر بيانات : قيمة متعددة ( لون ) ، ومشير يشير إلى العنصر التالى فى القائمة .

وعلى هذا ... فأول مكون له القيمة red ، والثانى green ، والثالث blue . وتتحدد بداية القائمة بمشير منفصل يسمى start ، كما تتحدد نهايتها بالقيمة NIL ( وسوف يذكر المزيد عن ذلك فيما بعد ) .

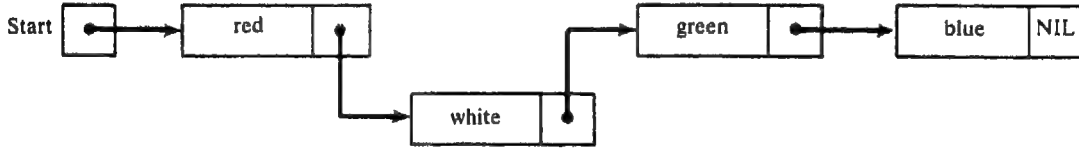
دعنا نضيف الآن مكونا جديدا قيمته white بين green ، red . لعمل ذلك ... فإننا نغير المشير كما هو موضح فى شكل (b) ١٣-١

(١) استخدم اسم مشير للدلالة على الكلمة الإنجليزية pointer ، وذلك لأنه سبق استخدام كلمة مؤشر للدلالة على الكلمة الإنجليزية parameter .

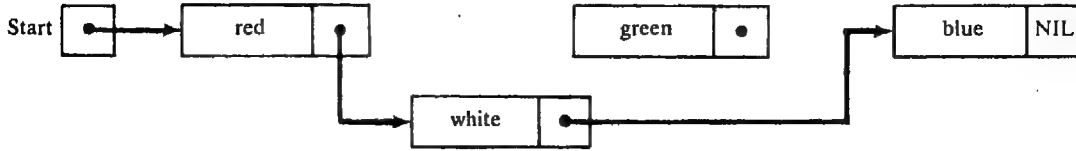
فإذا ما أردنا أن نحذف المكون الذي قيمته green فإننا نغير المشير ببساطة الذي يصاحب المكون الثاني كما هو موضح في شكل (c) ١٣-١ .



(ا)



(ب)



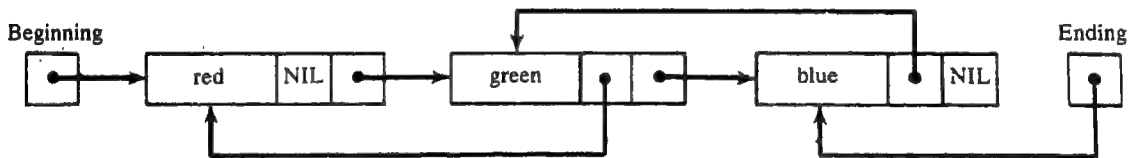
(ج)

شكل (١٣-١)

وهناك أنواع عديدة مختلفة من التكوينات المتصلة تشمل القوائم الخطية linear المتصلة (حيث تتصل كل مكوناتها معا بطريقة متسلسلة) . والقوائم المتصلة بعدة مشيرات (تسمح بالحركة للأمام وللخلف داخل القائمة) . والقوائم الدائرية circular (وهي قوائم خطية ليس لها بداية ونهاية) . والأشجار trees (والتي ترتب فيها المكونات ترتيبا هرميا) . وقد رأينا بالفعل توضيحا لاتصال خطي في مثال (١٣-١) . وفي المثال التالي يوجد بعض أنواع الاتصالات الأخرى .

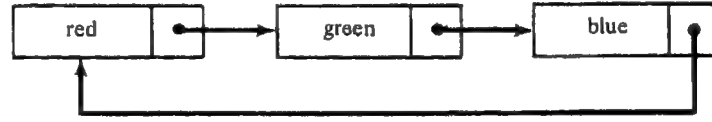
مثال (١٣-٢)

نرى في الشكل (١٣-٢) قائمة اتصال خطية تشبه القائمة الموجودة في شكل (a) ١٣-١ . والآن نرى على أية حال أن هناك مشيرين مصاحبين لكل مكون من المكونات ، وهما مشير أمامي ومشير خلفي . ويسمح هذان المشيران بالحركة داخل القائمة في كل من الاتجاهين ، أي من بدايتها وحتى نهايتها ، ومن نهايتها وحتى بدايتها .



شكل (١٣-٢)

اعتبر الآن شكل ١٣ - ٣ . هذه القائمة تشبه القائمة الموجودة في شكل ( a ) ١٣ - ٣ ، فيما عدا أن آخر عنصر بيانات ( blue ) يشير إلى أول عنصر بيانات ( red ) . وعلى هذا ... فهذه القائمة لا يوجد لها بداية ونهاية . ويشار إلى مثل هذه القائمة بأنها قائمة دائرية .

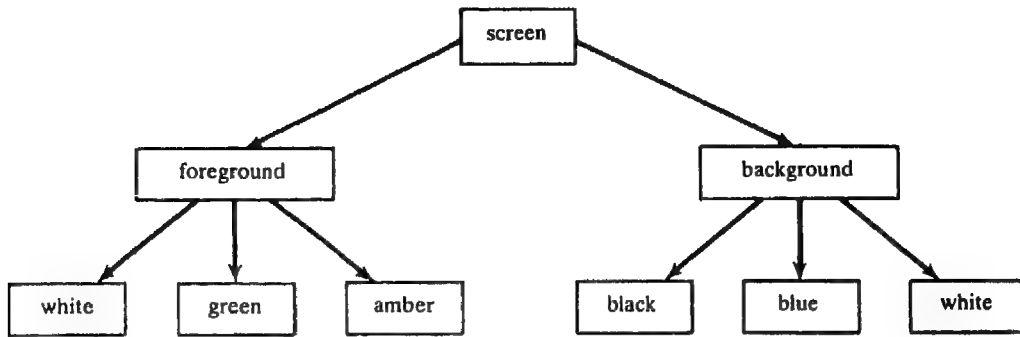


شكل ( ١٣ - ٣ )

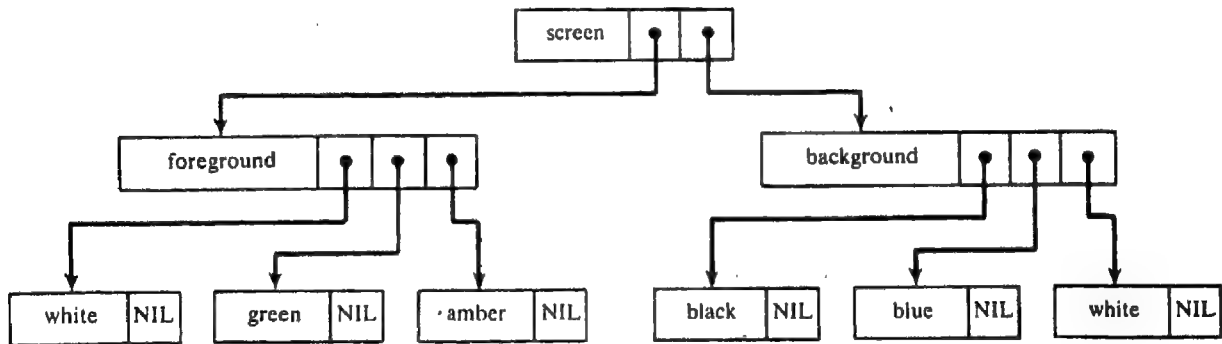
نرى أخيراً في شكل ( a ) ١٣ - ٤ مثالاً لشجرة . وتحتوي الشجرة على نقاط وأفرع مرتبة بطريقة هرمية تـ، تحدد التكوين الهرمي المناظر للبيانات ( والشجرة الثنائية binary tree هي شجرة تكون كل نقطة فيها لها أكثر من فرع واحد ) .

وفي شكل ( a ) ١٣ - ٤ نقطة الجذر لها القيمة screen ، والأفرع المصاحبة تقود إلى النقاط التي قيمها هي foreground ، background على التوالي . وبالمثل فالأفرع التي تصاحب foreground تعود إلى النقاط التي قيمها هي white ، green ، amber . والأفرع التي تصاحب background تعود إلى النقاط التي قيمها هي white ، blue ، black .

ويوضح شكل ( b ) ١٣ - ٤ الطريقة التي تستخدم بها المشيرات في أعداد الشجرة .



( ا )



( ب )

( شكل ١٣ - ٤ )

ولا يحتوى البسكال على كل مكون متصل كنوع بيانات مستقل ، بدلا من ذلك يستخدم نوع فردى للبيانات ، هو المشير الذى يسمح بإعداد أنواع مختلفة من التكوينات المتصلة ، طبقا لفئة محددة من القواعد .

## 2. TYPE DEFINITIONS

### ٢ - تعريفات النوع :

فى التطبيقات التى تحتوى على قوائم متصلة يجب أن يستخدم نوعان مختلفان من المتغيرات : متغيرات مشيرة pointer variables ( وهى متغيرات تشير قيمها إلى متغيرات أخرى ) ، ومتغيرات إشارة referenced variables ( وهى متغيرات " تشير إلى " ) . ويجب إعطاء اهتمام كبير لتعريف النوع الذى يصاحب كل نوع من أنواع المتغيرات . وهذا صحيح بصفة خاصة بالنسبة للنوع المشير ، pointer type أى تعريف النوع الذى يستخدم لتوضيح متغيرات من نوع المشير .

وبصفة عامة ... تعريف النوع المشير يكتب على النحو التالى :

`TYPE pointer type = ↑type identifier`

حيث يشير identifier بدون السهم إلى النوع المناظر لمتغير الإشارة . ( يظهر تعريف النوع هذا فى برنامج فيما بعد وذلك بعد تعريف نوع المشير ) .

والفكرة الأساسية من وراء تعريف النوع المشير هى ما يلى :

تشير قيمة المتغير من نوع المشير إلى بعض متغيرات أخرى تم تحديد نوعها بواسطة نوع المعرف . وهذه الفكرة موضحة فى شكل ١٣-٥ ، حيث نرى متغيراً من نوع المشير p تشير قيمته إلى متغير إشارة مناظر . وقيمة متغير الإشارة هذا يمثلها s . ومتغير الإشارة يكون من النوع المحدد بواسطة نوع المعرف identifier .



شكل (١٣-٥)

وعادة ما يعرف متغير الإشارة كسجل تحتوى حقوله على مكونات داخل القائمة المتصلة . وعادة مايكون آخر حقل عبارة عن عنصر بيانات من النوع المشير ، يحدد موقع المكون التالى . ويمثل هذا الحقل الاتصال بين المكون الحالى والمكون التالى له .

### مثال (١٣-٢)

اعتبر تعريفات النوع التالية :

```

TYPE primary = (red,green,blue);
pointer = ↑hue;
hue = RECORD
    color : primary;
    nextcolor : pointer
END;
    
```

يحدد السطر الأول أن red , green , blue هي عناصر بيانات من النوع primary . ويعرف pointer في السطر الثاني بأنه من نوع مشير يصاحب متغير إشارة من نوع hue . ( لاحظ أن متغير الإشارة هذا يستخدم لتمثيل مكون من مكونات قائمة الاتصال ) . أخيرا نعرف hue بأنه سجل يحتوى على حقلين ، هما color من نوع primary و nextcolor وهو مشير للمكون الثانى فى القائمة المتصلة .

لاحظ أن متغير الإشارة hue لم يعرف إلا بعد ظهوره فى تعريف المشير . وهذا يختلف عن تعريفات الأنواع الأخرى فى البسكال ، والتي يجب أن يعرف بها معرف قبل أن يكون ظهورها ممكنا فى أى تعريف نوع آخر .

وحيث إنه يمكن تعريف النوع المشير بالنسبة لنوع متغير الإشارة المناظر فقط ، فإننا نقول إن نوع المشير مرتبط بـ bound بنوع متغير الإشارة الذى يشير إليه .

### ٣ - توضيحات المتغيرات : 3. VARIABLE DECLARATIONS

توضح المتغيرات من نوع المشير بالطريقة التقليدية ، أى توضح كما يلى :

`VAR pointer name : type`

حيث يشير pointer name إلى متغير من نوع المشير ، و type هو نوع المشير . وينتج عن هذا التوضيح متغير ساكن أو استاتيكي ( تقليدى ) تشير قيمته إلى متغير إشارة ( أى إلى مكون داخل قائمة متصلة ) .

وعلى أية حال ... لاتوضح متغيرات الإشارة referenced variables بالطريقة التقليدية ، حيث إن هذه المتغيرات يتم تحديدها وإلغاؤها بطريقة ديناميكية ، أى مع تنفيذ البرنامج . وعلى هذا ... يجب توضيح متغيرات الإشارة داخل مشيرات البرنامج التى تحتوى على عبارات إجرائية . ومثل هذه التوضيحات تتم بطريقة مختلفة عن توضيحات المتغيرات التى سبق التعرض لها من قبل .

ولعمل متغير إشارة ، فإننا نستخدم الإجراء القياسى new بكتابة مايلى :

`new(pointer name)`

ينتج عن هذه العبارة متغير جديد ( وهو متغير إشارة ) يعرف نوعه فى تعريفات النوع الرسمى . والمشير الذى يظهر إسمه فى عبارة new يشير تلقائيا إلى هذا المتغير الجديد . ويكون لمتغير الإشارة نفس الاسم مثل المتغير المشير . وعلى أية حال ... يكتب متغير الإشارة مع وجود سهم رأسى بعد اسمه ، بحيث يمكن تمييزه عن المتغير المشير المناظر .

مثال (١٣-٤)

اعتبر التخطيط الهيكلى التالى لأحد برامج البسكال .

```
PROGRAM sample(input,output);
TYPE primary = (red,green,blue);
   pointer = ↑ hue;
   hue = RECORD
       color : primary;
       nextcolor : pointer
   END;
VAR foreground,background : pointer;
```

(تكملة البرنامج فى الصفحة التالية)

```
BEGIN (* main action block *)
.
.
new(foreground);
.
.
END.
```

تتكرر تعريفات النوع من مثال ١٣ - ٣ . ويلي تعريفات النوع هذه توضيح أن المتغيرين , background , foreground من النوع المشير .

لاحظ أن هذه توضيحات تقليدية للمتغيرات . وعلى هذا يكون كل من foreground , background متغيرين استاتيكيين يمكن تحديد قيمهما بالطريقة التقليدية .

ومن ناحية أخرى ... ينتج عن عبارة new الموجودة داخل المجموعة الاجرائية الرئيسية متغير إشارة من نوع hue يسمى foreground . وأكثر من ذلك .. فإن المتغير المشير المناظر foreground سوف يشير تلقائيا إلى foreground .

لاحظ أن متغير الإشارة foreground هو متغير من نوع السجل ، مكون من حقلين . وعلى هذا ... يمثل foreground , color قيمة متعددة من نوع primary ، ويمثل foreground , nextcolor مشيرا يمكن تحديده لمتغير الإشارة التالي :

#### ٤ - العمليات مع المتغيرات المشيرة ومتغيرات الإشارة

### 4. OPERATIONS WITH POINTER VARIABLES AND REFERENCED VARIABLES

هناك نوعان من العمليات يمكن إجراؤها مع المتغيرات المشيرة ومتغيرات الإشارة ، وهما التحديد Assignment والمقارنة Comparison . ويجب أن يجرى كل من نوعي العمليات مع عناصر متشابهة ، أى يجب أن تحدد المشيرات أخرى ، أو تقارن المشيرات بمشيريات أخرى ، وتقارن متغيرات الإشارة بمتغيرات إشارة أخرى .

دعنا نعتبر أولا عملية التحديد . افرض أن كلاً من p1 , p2 متغير مشير من نفس النوع . يمكننا على ذلك تحديد القيمة p1 للمتغير p2 بكتابة مايلي :

p2 := p1

وبالمثل افرض أن كلاً من p1 , p2 متغير إشارة يحتوى على حقل من نوع المشير يسمى next . يمكننا على ذلك كتابة مايلي

( أنظر الصفحة التالية ) .



```
p2 := p1↑.next
p2↑.next := p1
p2↑.next := p1↑.next
```

وهكذا ، وفى كل الحالات نحدد قيمة أحد المتغيرات المشيرة لمتغير مشير آخر ( بافتراض أن متغيرات الإشارة يتم إنتاجها فى كل حالة بالطبع ) .

ويمكن أيضا أن يشير المتغير إلى لاشئ وذلك بتحديد القيمة الخاصة NIL لهذا المتغير ( لاحظ أن NIL هى كلمة محجوزة من كلمات البسكال ) . وعلى هذا ... فإذا لم نرد أن يشير p1 إلى أى شئ ، فإننا نكتب مايلى :

```
p1 := NIL
```

وسوف نرى استخداما لهذه السمة فيما بعد فى هذا الفصل .

لاحظ أننا رأينا الآن طريقتين مختلفتين لتحديد قيم لمتغيرات مشيرة . إحدى الطريقتين هى استخدام عبارة تحديد تقليدية كما سبق ذكره . والطريقة الأخرى هى استخدام الإجراء القياسى new كما سبق ذكره فى القسم السابق . ( تذكر أن القيمة تتحدد تلقائيا لمتغير مشير عند إعداد متغير إشارة مناظر عن طريق عبارة new ) .

يمكن أيضا تحديد متغيرات إشارة إلى متغيرات إشارة أخرى ، على أن تكون جميعها من نفس النوع . وعلى ذلك ... يمكننا أن نكتب مايلى :

```
p2↑ := p1↑
```

ويتسبب ذلك فى تحديد قيمة كل حقل موجود داخل p1 للحقل المناظر له داخل p2 .

ويمكن إجراء مثل هذا التحديد مع حقول فردية لمتغيرات إشارة ، أو مع محتويات متغيرات إشارة .

مثال (١٣-٥)

اعتبر التوضيحات التالية ، والمكررة من المثال السابق .

```
TYPE primary = (red,green,blue);
pointer = ↑hue;
hue = RECORD
    color : primary;
    nextcolor : pointer
END;
VAR foreground,background : pointer;
```

إذا ما أعد متغيرا الإشارة foreground↑ , background↑ ( عن طريق عبارة new ) . فكل مايلى يكون على ذلك تحديدات مشيرة صحيحة .

```
foreground := background
background := NIL
foreground := background↑.nextcolor
foreground↑.nextcolor := background
foreground↑.nextcolor := background↑.nextcolor
```

كما يمكننا أن نحدد أيضا بعض أو كل قيم  $\text{background}^\uparrow$  ,  $\text{foreground}^\uparrow$  . وعلى هذا ... يمكننا أن نكتب مايلي :

```
background↑ := foreground↑
background↑.color := foreground↑.color
```

وأخيرا يمكننا أن نحدد قيمة متعددة لحقل من نفس النوع في أحد متغيرات الإشارة . فمثلا :

```
foreground↑.color := green
```

ويجب أن يفهم القارئ فهما كاملا الفرق بين التحديد الذي يشمل متغيرات مشيرة ، والتحديد الذي يشمل متغيرات إشارة . ولتوضيح هذه النقطة ، افرض أن المشيرين  $p1$  ,  $p2$  يشيران إلى متغيري إشارة مناظرين ، كما هو موضح في شكل ( a ) ١٣ - ٦

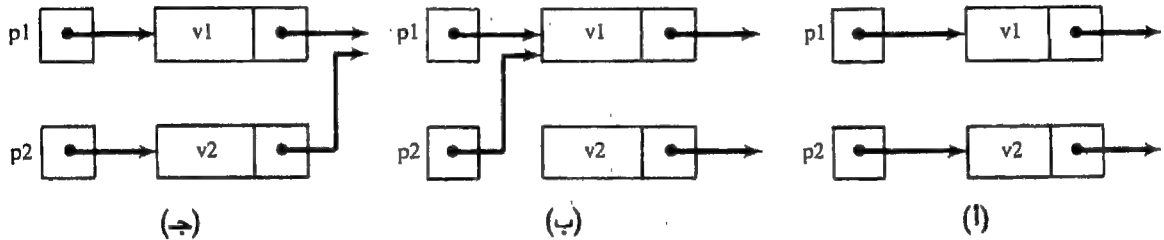
وعلى هذا ... فإن التحديد التالي :

```
p2 := p1
```

يتسبب في أن يشير كل من المشيرين إلى متغير الإشارة  $p1^\uparrow$  ، كما هو موضح في شكل ( b ) ١٣ - ٦ . ومن ناحية أخرى ... فإن التحديد :

```
p2↑ := p1↑
```

يتسبب في تحديد قيم متغير الإشارة  $p1^\uparrow$  لمتغير الإشارة  $p2^\uparrow$  . وهذا موضح في شكل ( c ) ١٣ - ٦ .



شكل ١٣ - ٦

ويمكن مقارنة المتغيرات المشيرة باستخدام مؤثرى علاقيين ، هما  $=$  ،  $<$  ،  $>$  . وتسمح هذه السمة لتحديد ما إذا كان متغيران مشيران لهما نفس القيمة أو لا ، أو إذا ما كان متغير مشير تحددت له القيمة NIL .

## مثال (١٣-٦)

اعتبر مرة أخرى التوضيحات المعطاة في الأمثلة ٤ - ١٣ و ٥ - ١٣ ، أى اعتبر مايلي :

```
TYPE primary = (red,green,blue);
pointer = ↑hue;
hue = RECORD
    color : primary;
    nextcolor : pointer
END;
VAR foreground,background : pointer;
```

فيما يلي عدة عبارات تحتوى على مقارنات بين متغيرات مشيرة background , foreground ( وتميل هذه الأمثلة لتوضيح التكوين فقط ، وليس لها أى معنى منطقي خاص ) .

```
IF foreground = background THEN foreground := NIL
```

```
WHILE foreground <> background DO
```

```
REPEAT
```

```
·
·
·
```

```
UNTIL background = foreground↑.nextcolor
```

```
IF background = NIL THEN foreground↑.color := red
```

ويمكن أيضا مقارنة متغيرات الإشارة ، على أن تكون من نفس النوع . ويمكن أن تشمل مثل هذه المقارنات محتويات متغيرات أو حقول فردية . فإذا ما قورنت حقول فردية ، فيجب أن تكون من نفس النوع أيضا .

## مثال (١٣-٧)

فيما يلي بعض أمثلة لمقارنة متغيرات إشارة أو حقول فردية داخل متغيرات إشارة . وتستخدم مرة أخرى التوضيحات المقدمة في الأمثلة السابقة ، أى :

```
TYPE primary = (red,green,blue);
pointer = ↑hue;
hue = RECORD
    color : primary;
    nextcolor : pointer
END;
VAR foreground,background : pointer;
```

وفيما يلي بعض المقارنات الصحيحة .

```
IF foreground↑ = background↑ THEN foreground := NIL
```

```
IF foreground↑.color <> blue THEN foreground↑.color := blue
```

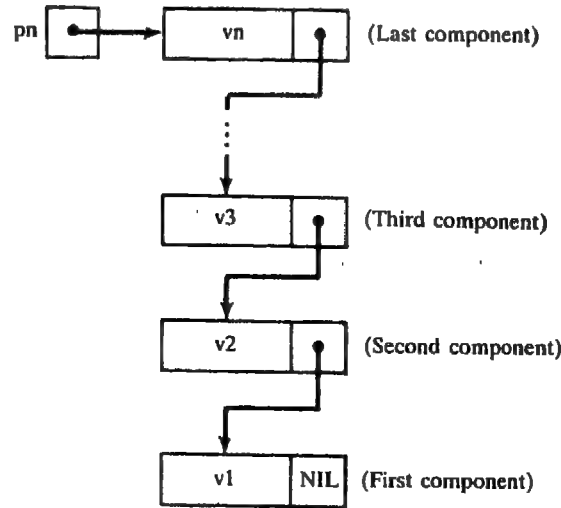
```
WHILE background↑.color <> foreground↑.color DO
```

## ٥ - إنتاج وإلغاء متغيرات ديناميكية

## 5. CREATING AND DESTROYING DYNAMIC VARIABLES

نعيد انتباهنا الآن إلى بعض المواقف التقليدية للبرمجة ، والتي تتطلب متغيرات مشيرة وما يناظرها من متغيرات إشارة . سوف نرى أن استخدام المؤشرات المشيرة يسمح بإنتاج وإلغاء متغيرات إشارة ديناميكياً ، أى مع تنفيذ البرنامج .

ولكى نوضح طريقة إنتاج وحذف متغيرات إشارة ديناميكياً ، فإننا نعمل مع النوع المعتاد من القائمة المتصلة ، التي يشير كل مكون من مكوناتها إلى المكون السابق له . وعلى هذا ... يشير المكون الثانى إلى المكون الأول ، ويشير الثالث إلى الثانى . وهكذا كما هو موضح فى شكل ١٣ - ٧ . ويسمى هذا المخطط « الأخير فى دخوله هو الأول فى خروجه » (LIFO) ، last-in , first-out ، حيث إن آخر مكون يتم إدخاله فى القائمة ، يكون أول مكون يغادرها .



شكل ١٣ - ٧

ويمكن إنتاج القائمة المتصلة بقراءة كل مكون جديد داخلها ، ثم توصيل هذه المكون بالمكون السابق له .

افترض على سبيل المثال أن item هو متغير من النوع المشير ، وأنه يناظر متغير إشارة  $\uparrow item$  يحتوى على حقلين ، هما حقل سلسلة يسمى  $\uparrow item - name$  ، وحقل مشير اسمه  $\uparrow item, next$  . ويقدم المشير اتصالاً للمكون الذى سبق إدخاله ، والذي يتحدد موقعه بواسطة متغير مشير pointer .

ولكى يتم إنتاج أول مكون ، فإننا ننتج متغيراً ديناميكياً ( item ) يقرأ قيمة name ،  $\uparrow item$  فى القائمة ، ويضع قيمة NIL للمتغير  $\uparrow item, next$  . ويمثل هذا المكون الموجود فى قاعدة القائمة ، ثم يعد المتغير المشير ليشير إلى هذا المكون ( بحيث يعكس تعريف المكون السابق عندما يتم إدخال المكون الثانى ) .

وعلى هذا ... ينتج المكون الأول بكتابة مايلى :

```
new(item);
readname(item↑.name);
item↑.next := NIL;
pointer := item;
```

حيث read name هو إجراء يقرأ عناصر item↑.name رمزا رمزا .

وتعتمد هذه الخطوات لكل مكون تالٍ ، فيما عدا أن item↑.next يعد ليشير إلى المكون السابق ، وذلك بتحديد القيمة الحالية لـ pointer . وعلى هذا :

```
new(item);
readname(item↑.name);
item↑.next := pointer;
pointer := item;
```

يجب وضع مجموعة العبارات هذه داخل إحدى أنواع الدورات ( مثل مكون DO - WHILE ) ، بحيث يمكن تكراره لكل مكون تالٍ في القائمة .

ويمكن عرض القائمة ، وذلك بحركتها من القمة إلى القاعدة ( آخر مكون أولا ) . وفي البداية يحدد pointer موقعاً آخر ( أحدث ) مكوناً . وعلى هذا ... يمكننا أن نعرض محتويات القائمة ببساطة ، وذلك بتكرار الثلاث عبارات التالية لكل مكون من مكونات القائمة .

```
item := pointer;
writeln(item↑.name);
pointer := item↑.next;
```

ويجب أن توضع هذه العبارات داخل دورة ، يستمر تنفيذها حتى تصبح قيمة pointer هي NIL .

### مثال (١٣-٨)

إنتاج قائمة منفصلة : فيما يلي برنامجاً كاملاً بلغة البيسكال ، ينتج ويعرض قائمة متصلة طبقاً لمبدأ آخر عنصر يتم إدخاله يخرج أولاً LIFO ، وذلك باستخدام الطريقة التي سبق ذكرها .

```
PROGRAM makelist(input,output);

(* THIS PROGRAM USES POINTERS TO CREATE A LINKED LIST OF NAMES *)

TYPE link = ↑personal;
personal = RECORD
    name : PACKED ARRAY [1..40] OF char;
    next : link
END;
VAR item,pointer : link;

PROCEDURE readname(VAR newname : link);
(* this procedure reads a name into the computer *)
VAR count : 0..40;
BEGIN
    FOR count := 1 TO 40 DO newname↑.name[count] := ' ';
    write('New name: ');
    count := 0;
    WHILE NOT eoln DO
        BEGIN
            count := count + 1;
            read(newname↑.name[count])
        END;
    END;
```

(تكملة البرنامج في الصفحة التالية)

```

readln
END;

BEGIN (* main action block *)

  BEGIN (* create the list *)
    new(item);
    readname(item);
    item↑.next := NIL;
    pointer := item;
    WHILE NOT ((item↑.name[1] IN ['E','e'])
      AND (item↑.name[2] IN ['N','n'])
      AND (item↑.name[3] IN ['D','d'])) DO
      BEGIN
        new(item);
        readname(item);
        item↑.next := pointer;
        pointer := item
      END;
      pointer := item↑.next
    END; (* create list *)

  BEGIN (* display the list *)
    writeln;
    WHILE pointer <> NIL DO
      BEGIN
        item := pointer;
        writeln(item↑.name);
        pointer := item↑.next
      END;
    END (* display list *)
  END.

```

يحتوى هذا البرنامج على إجراء اسمه readname يستخدم فى قراءة اسم يحتوى على عدد من الخانات ، يصل إلى ٤٠ خانة . ويبدأ الإجراء بتحديد مواقع فارغة لكل خانات الاسم . وبعد ذلك يتم إدخال الاسم الفعلى داخل الكمبيوتر رمزا رمزا ، حتى تكتشف نهاية السطر ( eoln ) والتي تتسبب فى إرجاع العربة .

لاحظ أن readname يستخدم مؤشراً متغيراً اسمه newname من نوع link . وعلى هذا ... فإن المؤشر عبارة عن متغير من نوع المشير . والقيمة التى تعود بواسطة هذا الإجراء ليست مشيراً على أية حال ، لكنها إحدى العناصر الموجودة داخل متغير الإشارة المناظر ( المسمى name . name↑ ) . وعلى ذلك ... فإننا نرى أن متغير الإشارة يمكن استغلاله داخل إجراء عن طريق نقل المتغير من النوع المشير المصاحب له إلى ( أو من ) الإجراء .

وتحتوى المجموعة الإجرائية الأساسية على عبارتين مركبتين ، إحداهما تنتج القائمة المتصلة ، والأخرى تعرضها على التوالى . والمنطق المستخدم داخل كل من هاتين العبارتين مباشر ، وذلك فى ضوء المناقشة التى سبقت هذا المثال . لاحظ على أية حال دورة WHILE - DO الموجودة فى أول عبارة . تتسبب هذه الدورة فى تكرار إدخال الأسماء الجديدة ، حتى يتم إدخال الكلمة end كاسم جديد ( سواء أكانت مكتوبة بحروف كبيرة أم حروف صغيرة ) . لاحظ أيضاً أن المتغير من نوع المشير item يمرر إلى الإجراء readname .

افرض الآن أن البرنامج يستخدم لإنتاج قائمة متصلة تحتوى على الأسماء التالية :

Sharon, Gail, Susan, Marla, Marc, Amy, Megan

فيما يلي حوار المدخلات والمدخلات ، مع وضع خط تحت استجابة المستفيد .

New name: Sharon  
 New name: Gail  
 New name: Susan  
 New name: Marla  
 New name: Marc  
 New name: Amy  
 New name: Megan  
 New name: end

Megan  
 Amy  
 Marc  
 Marla  
 Susan  
 Gail  
 Sharon

لاحظ أن الأسماء مطبوعة في ترتيب عكسى ، أى أن آخر اسم تم إدخاله طبع أولا .

ويمكن إدخال أى عنصر إضافى داخل بعض المواقع المحددة داخل القائمة المتصلة التى تم إعدادها . ولعمل ذلك يجب علينا أن ندخل العنصر الجديد ، محددين موقعه داخل القائمة ، ثم نضبط بعض المشيرات .

افرض - بصفة خاصة - أن item يمثل العنصر المراد إضافته ، وأن pointer يحدد العنصر الموجود خلف العنصر الجديد . تسمح العبارتان التاليتان بإدخال العنصر الجديد ( مفترضا أنه name . item مرة أخرى ) .

```
new(item);
readname(item↑.name);
( يمكن تحديد ) الموقع المحدد للعنصر الجديد بكتابة مايلى :
```

```
readname(itemname);
pointer := last;
WHILE (pointer↑.name <> itemname) AND
  (pointer↑.next <> NIL) DO pointer := pointer↑.next;
```

حيث يكون item name متغير سلسلة يمثل العنصر التالى للعنصر الجديد ، ويكون last مشيرا لآخر عنصر داخل القائمة .

أخيرا يحدث تضبيط المشيرات كما يلى ، وذلك إذا ما كان يراد إدخال عنصر جديد فى نهاية القائمة .

```
item↑.next := last;
last := item;
```

والا فيكتب :

```
item↑.next := pointer↑.next;
pointer↑.next := item;
```

في كل حالة من هذه الحالات يوجه جزء المشير للعنصر الجديد إلى العنصر الذي يسبقه ، والعنصر ( القديم ) الذي يلي العنصر الذي تمت إضافته يتم تضبيطه .

وحذف أحد العناصر يتم بطريقة تشبه إضافة عنصر جديد . فيتم الحذف الفعلي باستخدام عبارة dispose ، أى باستخدام .

`dispose(pointer name)`

وهذه العبارة متماثلة مع عبارة new . وعلى هذا ... فإن dispose هو إجراء قياسي يقبل متغيراً من النوع المشير كمؤشر . ويتسبب استخدام هذا الإجراء في إلغاء متغير الإشارة المصاحب للمشير المعطى . وأى إشارة تالية لهذا المتغير ( عن طريق مشير آخر ) تكون عند ذلك غير معرفة .

والطريقة الشاملة هي كما يلي :

نحدد أولاً العنصر المراد حذفه ، ثم نعيد ضبط المشيرات ، وأخيراً نجرى dispose على العنصر المراد إزالته . وعلى هذا ... يكتب مايلي :

```
readname(itemname);
IF last↑.name = itemname
  THEN BEGIN (* delete last item *)
    item := last;
    last := item↑.next;
    dispose(item)
  END
ELSE BEGIN (* find item, then delete *)
  pointer := last;
  item := last↑.next;
  WHILE (item↑.name <> itemname) AND
    (item↑.next <> NIL) DO
    BEGIN
      pointer := item;
      item := item↑.next
    END;
  IF item↑.name = itemname THEN
    BEGIN
      pointer↑.next := item↑.next;
      dispose(item)
    END
  END;
END;
```

يمكن لهذه العبارة أن تحقق موقفين مختلفين : حذف آخر عنصر ، وحذف بعض العناصر الأخرى . فإذا ما أريد حذف آخر عنصر ، فيتم ضبط المشير إلى نهاية القائمة ( last ) ، بحيث يشير إلى العنصر التالي في القائمة ،



أى أن :

```
last := item↑.next;
```

أما إذا ما أريد حذف بعض العناصر الأخرى ، فيجعل العنصر التالى للعنصر الذى يحذف مشيراً إلى العنصر التالى له فى القائمة ، أى أن :

```
pointer↑.next := item↑.next;
```

مثال (١٣-٩)

تشغيل قائمة متصلة . نوسع فى هذا المثال بعض الأفكار التى سبق تقديمها فى مثال ١٣-٨ . ويصفى خاصية نقدم الآن برنامجاً متداخلاً بلغة البسكال ، يسمح لنا بإنتاج قائمة متصلة ، ويضيف عنصراً جديداً ويحذف أى عنصر . والبرنامج يستخدم القوائم لتسهيل استخدامه بواسطة مستفيدين غير ملمين بالبرمجة . ويوجد به جزء لعرض القائمة بترتيب عكسى ( أى يظهر آخر عنصر أولاً ) بعد القيام باختيار أى عنصر من القائمة .

ويبدأ البرنامج بتعريفات النوع ، وتوضيحات المتغيرات التالية :

```
TYPE line = PACKED ARRAY [1..40] OF char;
link = ↑personal;
personal = RECORD
    name : line;
    next : link
END;
VAR item,pointer,last : link;
choice : 1..4;
count : 0..40;
itemname : line;
```

وتشبه هذه التعريفات والتوضيحات مثيلاتها المقدمة فى مثال ١٣-٧ ، بالرغم من أننا توسعنا فيها بعض الشئ لتشمل السمات الإضافية فى البرنامج الحالى .

اعتبر المجموعة الرئيسية الآن . إذا وضعنا إنتاج القائمة والعبارات الإجرائية المصاحبة لكل اختيار من اختيارات القائمة داخل إجراءات منفصلة . وأن المجموعة الإجرائية الرئيسية لا تحتوى إلا على مجموعة إشارات لهذه الإجراءات فى معظم تكوينها . وعلى هذا ... يمكننا أن نكتب ما يلى :

```
BEGIN (* main action block *)
REPEAT
    menu;
CASE choice OF
    1 : create;
    2 : add;
    3 : delete;
    4 :
END
UNTIL choice = 4
END.
```

لاحظ أننا اتصلنا أولاً بالإجراء menu الذى يتسبب فى إدخال قيمة choice من لوحة المفاتيح ( يمكن أن يأخذ choice القيم الصحيحة ١ أو ٢ أو ٣ أو ٤ ) . نتصل بعد ذلك بإجراء إضافي يعتمد اختياره على القيمة التى تم تحديدها choice . وتتكرر هذه العملية حتى تتحدد قيمة لـ choice ، والتي تحدد توقف الشرط .

والآن دعنا نفحص الإجراء menu . إنه إجراء بسيط جداً ، يحتوى على عبارات مدخلات ومخرجات فقط .

```
PROCEDURE menu;

(* this procedure displays the main menu *)

BEGIN
    page;
    writeln('Main menu:');
    writeln;
    writeln(' 1 - Create the linked list');
    writeln;
    writeln(' 2 - Add a component');
    writeln;
    writeln(' 3 - Delete a component');
    writeln;
    writeln(' 4 - End');
    writeln;
    write('Enter your choice (1, 2, 3 or 4) -> ');
    readln(choice);
    writeln
END;
```

( يمكن وضع هذه العبارات ببساطة داخل المجموعة الإجرائية الرئيسية . وقد وضعت فى إجراء مستقل لتبسيط التنظيم الشامل لمنطق البرنامج ) .

والإجراءات المستخدمة فى إنتاج قائمة جديدة ، وإدخال اسم من لوحة المفاتيح ، وعرض القائمة الحالية تستخدم نفس المنطق بالضرورة مثل برنامج البسكال المعطى فى مثال ١٢-٨ . ( لاحظ على أية حال أن مجموعات العبارات المطلوبة لإنتاج القائمة وعرضها نقلت من المجموعة الإجرائية الرئيسية إلى برامج فرعية subroutines فردية ) وعلى هذا ... يمكن إنتاج القائمة بالاتصال بالإجراء التالى :

```
PROCEDURE create;

(* this procedure creates the linked list

    item    -> the new (most recent) component
    pointer -> the preceding component

    ('item' points to 'pointer') *)

BEGIN
    new(item);
    write('New name: ');
    readname(item↑.name);
    item↑.next := NIL;
    pointer := item;
    WHILE NOT ((item↑.name[1] IN ['E','e'])
                AND (item↑.name[2] IN ['N','n'])
                AND (item↑.name[3] IN ['D','d'])) DO
```

(تكملة البرنامج فى الصفحة التالية)

```

        BEGIN
            new(item);
            write('New name: ');
            readname(item^.name);
            item^.next := pointer;
            pointer := item
        END;
        last := item^.next;
        display
    END;

```

لاحظ أن آخر عنصر في القائمة ملحق به المشير last . يمكننا ذلك من إيجاد نهاية القائمة عندما نرغب في عرضها ، وإضافة عنصر جديد ، أو حذف أحد عناصرها .

لاحظ أيضا أن القائمة الجديدة تعرض تلقائيا بعد إنتاجها .

الإجراء readname الذي يتم الاتصال به بواسطة الإجراء سالف الذكر ( وإجراءات أخرى أيضا ) . يكتب على النحو التالي :

```

PROCEDURE readname(VAR name ; line);

(* this procedure reads a name into the computer *)

BEGIN
    FOR count := 1 TO 40 DO name[count] := ' ';
    count := 0;
    WHILE NOT eoln DO
        BEGIN
            count := count + 1;
            read(name[count])
        END;
    readln
END;

```

الاجراء في هذا المثال يشبه تماما الإجراء الذي له نفس الاسم الموجود في مثال ١٣ - ٨ . لاحظ أن الإجراء الحالي يستخدم على أية حال متغير سلسلة كمؤشر من نوع المتغير ، بينما الصيغة السابقة تستخدم مشيرا كمؤشر متغير . وتقدم المعالجة الحالية لمؤشر المتغير عمومية أكثر . وعلى هذا ... فهي تسمح بالاتصال بهذا الإجراء عن طريق عدة إجراءات أخرى ( وهي بالاسم create , add , delete ) .

والإجراء المستخدم لعرض القائمة يمكن أن يكتب على النحو التالي :

```

PROCEDURE display;

(* this procedure displays the complete list

item    -> the current component to be displayed
pointer -> the preceding component

('item' points to 'pointer') *)

```

```
BEGIN
  writeln;
  pointer := last;
  WHILE pointer <> NIL DO
    BEGIN
      item := pointer;
      writeln(item^.name);
      pointer := item^.next;
    END
  END;
```

يبدأ هذا الإجراء بتحديد نهاية القائمة ( Pointer := Last ) ثم يستمر بنفس الطريقة مثل البرنامج الموضح في مثال ١٣-٨ .

ويسمح الإجراء التالي بإضافة عنصر إلى القائمة المتصلة التي تم إنتاجها .

```
PROCEDURE add;
(* this procedure allows one component
   to be added to the linked list
   item    -> the component to be added
   pointer -> component below (i.e., pointing to) the new component
              ('pointer' points to 'item'). *)
BEGIN
  new(item);
  writeln;
  write('New name: ');
  readname(item^.name);
  write('Place ahead of: (press RETURN if new item is last) ');
  readname(itemname);
  IF itemname[1] = ' '
    THEN BEGIN (* insert at end *)
      item^.next := last;
      last := item;
    END
    ELSE BEGIN (* find location, then insert *)
      pointer := last;
      WHILE (pointer^.name <> itemname) AND
        (pointer^.next <> NIL) DO pointer := pointer^.next;
      item^.next := pointer^.next;
      pointer^.next := item;
    END;
  display;
END;
```

يتسبب هذا الإجراء أولاً في إدخال الاسم الجديد ( من لوحة المفاتيح ) ، وبعد ذلك تحديد موقعه . وبعد ذلك يتبع الإجراء المنطق الذي سبق ذكره في هذا الفصل ، والذي يسمح بإدخال عنصر جديد في نهاية القائمة ( أى عند قمتها ) أو في أى مكان آخر مناسب داخل القائمة . لاحظ أن القائمة الجديدة تعرض تلقائياً بعد إتمام الإضافة .

اعتبر الآن الإجراء المستخدم في حذف عنصر من مكان محدد داخل القائمة . يمكن كتابة هذا الإجراء على النحو التالي : ( أنظر الصفحة القادمة ) .

```

PROCEDURE delete;

(* this procedure causes one component
   to be deleted from the linked list

   item    -> the component to be deleted
   pointer -> component above (i.e., pointing to)
               the component to be deleted

               ('pointer' points to 'item')      *)

BEGIN
  writeln;
  write ('Name to be deleted: ');
  readname(itemname);
  IF last↑.name = itemname
  THEN BEGIN (* delete last item *)
    item := last;
    last := item↑.next;
    dispose(item)
  END
  ELSE BEGIN (* find item, then delete *)
    pointer := last;
    item := last↑.next;
    WHILE (item↑.name <> itemname) AND
      (item↑.next <> NIL) DO
      BEGIN
        pointer := item;
        item := item↑.next;
      END;
    IF item↑.name = itemname THEN
      BEGIN
        pointer↑.next := item↑.next;
        dispose(item)
      END
    END;
  display
END;

```

لقد سبق ظهور التفاصيل المنطقية لهذا الإجراء في هذا الفصل . لاحظ أن القائمة الجديدة تعرض تلقائياً بعد إتمام عملية الحذف .

وفيما يلي البرنامج كاملاً .

```

PROGRAM linklist(input,output);

(* THIS IS A MENU-DRIVEN PROGRAM THAT USES POINTERS
   TO PROCESS A LINKED LIST OF NAMES *)

TYPE line = PACKED ARRAY [1..40] OF char;
link = ↑personal;
personal = RECORD
  name : line;
  next : link
END;

```

(تكملة البرنامج في الصفحة التالية)

```

VAR item,pointer,last : link;
    choice : 1..4;
    count : 0..40;
    itemname : line;

PROCEDURE readname(VAR name : line);

(* this procedure reads a name into the computer *)

BEGIN
    FOR count := 1 TO 40 DO name[count] := ' ';
    count := 0;
    WHILE NOT eoln DO
        BEGIN
            count := count + 1;
            read(name[count])
        END;
    readln
END;

PROCEDURE display;

(* this procedure displays the complete list

    item    -> the current component to be displayed
    pointer -> the preceding component

    ('item' points to 'pointer') *)

BEGIN
    writeln;
    pointer := last;
    WHILE pointer <> NIL DO
        BEGIN
            item := pointer;
            writeln(item^.name);
            pointer := item^.next
        END
    END;

PROCEDURE create;

(* this procedure creates the linked list

    item    -> the new (most recent) component
    pointer -> the preceding component

    ('item' points to 'pointer') *)

BEGIN
    new(item);
    write('New name: ');
    readname(item^.name);
    item^.next := NIL;
    pointer := item;
    WHILE NOT ((item^.name[1] IN ['E','e'])
        AND (item^.name[2] IN ['N','n'])
        AND (item^.name[3] IN ['D','d'])) DO
        (تكملة البرنامج في الصفحة التالية)

```

```

        BEGIN
            new(item);
            write('New name: ');
            readname(item↑.name);
            item↑.next := pointer;
            pointer := item
        END;
        last := item↑.next;
        display
    END;

    PROCEDURE add;

    (* this procedure allows one component
       to be added to the linked list

       item    -> the component to be added
       pointer -> component below (i.e., pointing to) the new component
                  ('pointer' points to 'item') *)

    BEGIN
        new(item);
        writeln;
        write('New name: ');
        readname(item↑.name);
        write('Place ahead of: (press RETURN if new item is last) ');
        readname(itemname);
        IF itemname[1] = ' '
            THEN BEGIN (* insert at end *)
                    item↑.next := last;
                    last := item
                END
            ELSE BEGIN (* find location, then insert *)
                    pointer := last;
                    WHILE (pointer↑.name <> itemname) AND
                        (pointer↑.next <> NIL) DO pointer := pointer↑.next;
                    item↑.next := pointer↑.next;
                    pointer↑.next := item
                END;
        display
    END;

    PROCEDURE delete;

    (* this procedure causes one component
       to be deleted from the linked list

       item    -> the component to be deleted
       pointer -> component above (i.e., pointing to)
                  the component to be deleted
                  ('pointer' points to 'item') *)

    BEGIN
        writeln;
        write ('Name to be deleted: ');
        readname(itemname);
        IF last↑.name = itemname
            THEN BEGIN (* delete last item *)
                    item := last;
                    last := item↑.next;
                    dispose(item)
                END
    END

```

(تكملة البرنامج في الصفحة التالية)

```

ELSE BEGIN (* find item, then delete *)
    pointer := last;
    item := last↑.next;
    WHILE (item↑.name <> itemname) AND
        (item↑.next <> NIL) DO
        BEGIN
            pointer := item;
            item := item↑.next;
        END;
    IF item↑.name = itemname THEN
        BEGIN
            pointer↑.next := item↑.next;
            dispose(item)
        END
    END;
display
END;

PROCEDURE menu;

(* this procedure displays the main menu *)
BEGIN
    page;
    writeln('Main menu:');
    writeln;
    writeln(' 1 - Create the linked list');
    writeln;
    writeln(' 2 - Add a component');
    writeln;
    writeln(' 3 - Delete a component');
    writeln;
    writeln(' 4 - End');
    writeln;
    write('Enter your choice (1, 2, 3 or 4) -> ');
    readln(choice);
    writeln
END;

BEGIN (* main action block *)
    REPEAT
        menu;
        CASE choice OF
            1 : create;
            2 : add;
            3 : delete;
            4 :
        END
    UNTIL choice = 4
END.

```



دعنا نستخدم هذا البرنامج في إنتاج قائمة متصلة تحتوى على المدن التالية : Denver , Chicago , Boston , SanFrancisco , Pittsburgh , New Yourk . وبعد ذلك نضيف عدة مدن إضافية ، ونحذف عدة مدن أخرى ، موضحين كل سمات البرنامج . وسوف نحفظ قائمة المدن في ترتيب أبجدي في هذا التمرين . وعى هذا ... فإننا ندخل القائمة الاولى من الخلف ، بحيث تطبع المدن بالترتيب الصحيح ( لاحظ أن SanFrancisco ستكون في واقع الأمر في بداية القائمة ، وأن Boston ستكون في النهاية ) .

وفيما يلي الجزء المتداخل ، وكالعادة موضوع خط تحت استجابات المستفيد .

Main menu:

- 1 - Create the linked list
- 2 - Add a component
- 3 - Delete a component
- 4 - End

Enter your choice (1, 2, 3 or 4) -> 1

New name: San Francisco  
 New name: Pittsburgh  
 New name: New York  
 New name: Denver  
 New name: Chicago  
 New name: Boston  
 New name: end

Boston  
 Chicago  
 Denver  
 New York  
 Pittsburgh  
 San Francisco

Main menu:

- 1 - Create the linked list
- 2 - Add a component
- 3 - Delete a component
- 4 - End

Enter your choice (1, 2, 3 or 4) -> 2

(تكلمة البرنامج في الصفحة التالية)

New name: Atlanta

Enter ahead of (press RETURN if last item):

Atlanta  
Boston  
Chicago  
Denver  
New York  
Pittsburgh  
San Francisco

Main menu:

- 1 - Create the linked list
- 2 - Add a component
- 3 - Delete a component
- 4 - End

Enter your choice (1, 2, 3 or 4) -> 2

New name: Seattle

Enter ahead of (press RETURN if last item): San Francisco

Atlanta  
Boston  
Chicago  
Denver  
New York  
Pittsburgh  
San Francisco  
Seattle

Main menu:

- 1 - Create the linked list
- 2 - Add a component
- 3 - Delete a component
- 4 - End

Enter your choice (1, 2, 3 or 4) -> 3

Name to be deleted: New York

Atlanta  
Boston  
Chicago  
Denver  
Pittsburgh  
San Francisco  
Seattle

(تكملة البرنامج في الصفحة التالية)

Main menu:

- 1 - Create the linked list
- 2 - Add a component
- 3 - Delete a component
- 4 - End

Enter your choice (1, 2, 3 or 4) -> 2

New name: Washington

Enter ahead of (press RETURN if last item): Seattle

Atlanta  
Boston  
Chicago  
Denver  
Pittsburgh  
San Francisco  
Seattle  
Washington

Main menu:

- 1 - Create the linked list
- 2 - Add a component
- 3 - Delete a component
- 4 - End

Enter your choice (1, 2, 3 or 4) -> 3

Name to be deleted: Atlanta

Boston  
Chicago  
Denver  
Pittsburgh  
San Francisco  
Seattle  
Washington

Main menu:

- 1 - Create the linked list
- 2 - Add a component
- 3 - Delete a component
- 4 - End

Enter your choice (1, 2, 3 or 4) -> 2

(تكملة البرنامج في الصفحة التالية)

New name: Dallas  
Enter ahead of (press RETURN if last item): Chicago

Boston  
Chicago  
Dallas  
Denver  
Pittsburgh  
San Francisco  
Seattle  
Washington

Main menu:

- 1 - Create the linked list
- 2 - Add a component
- 3 - Delete a component
- 4 - End

Enter your choice (1, 2, 3 or 4) -> 3

Name to be deleted: Washington

Boston  
Chicago  
Dallas  
Denver  
Pittsburgh  
San Francisco  
Seattle

Main menu:

- 1 - Create the linked list
- 2 - Add a component
- 3 - Delete a component
- 4 - End

Enter your choice (1, 2, 3 or 4) -> 4

## Review Questions

## أسئلة مراجعة :

- (١) ماذا تعنى القائمة المتصلة ؟
- (٢) ماهو المشير ؟ وماهى العلاقة بين المشيريات والقوائم المتصلة ؟
- (٣) كيف تختلف القائمة المتصلة عن القائمة التتابعية - على سبيل المثال - الموجودة فى منظومة ذات بعد واحد ؟
- (٤) كيف تعرف بداية القائمة المتصلة ؟
- (٥) ماهى القائمة المتصلة الخطية ؟ (ب) وماهى القائمة المتصلة الدائرية ؟ وكيف تختلف القائمة المتصلة الدائرية عن القائمة المتصلة الخطية ؟
- (٦) ماهى الشجرة ؟ كيف تختلف الأشجار عن القوائم المتصلة الخطية ؟ (ب) وماهى الشجرة الثنائية ؟ وكيف تختلف الشجرة الثنائية عن بقية الأنواع الأخرى من الأشجار ؟
- (٧) ماهى النقاط والفروع ؟ وأى نوع من أنواع تكوينات البيانات الذى تصاحبه النقاط والفروع ؟
- (٨) ماهو نوع البيانات المستخدمة فى معالجة التكوينات المتصلة فى البسكال ؟
- (٩) كيف يعرف النوع المشير فى البسكال ؟
- (١٠) ماهو نوع تكوينات البيانات المستخدم بصفة عامة فى تعريف متغير إشارة ؟ وماسبب هذا الاختيار ؟
- (١١) صف الطريقة التى ترتبط بها أنواع المشيريات مع أنواع متغيرات إشارة كل منهما مع الآخر .
- (١٢) كيف توضح المتغيرات من النوع المشير ؟ وفى أى جزء من أجزاء برنامج البسكال تظهر هذه التوضيحات ؟
- (١٣) حدد فى كلمات محددة ، كيف توضح متغيرات الإشارة ، وكيف يختلف مفهوم توضيحات متغيرات الإشارة عن توضيحات الأنواع الأخرى للمتغيرات ؟
- (١٤) فى أى جزء من أجزاء برنامج البسكال يظهر توضيح متغيرات الإشارة ؟ قارن ذلك مع توضيحات أنواع المتغيرات الأخرى ، وحدد أسباب هذه الاختلافات .
- (١٥) ماهو الغرض من عبارة now ؟ وماهو نوع المؤشر المطلوب ؟
- (١٦) وضح العلاقة الموجودة بين متغير إشارة ، والمتغير المشير المناظر فيه .
- (١٧) كيف يمكن تمييز اسم متغير الإشارة عن اسم المتغير المشير المناظر له ؟
- (١٨) لخص القواعد المستخدمة فى تحديد متغير مشير لآخر .
- (١٩) كيف يمكن جعل متغير مشير يشير إلى لاشئ ؟

- (٢٠) صف طريقتين مختلفتين لتحديد قيم لمتغيرات مشيرة . ماهو الغرض من كل نوع من نوعي التحديد هذين ؟
- (٢١) لخص القواعد المستخدمة في تحديد متغير إشارة لآخر .
- (٢٢) هل يمكن تحديد حقول فردية موجودة داخل متغيرات إشارة لحقول أخرى ؟ ماهي القيود الموضوعة على هذا النوع من أنواع التحديد ؟
- (٢٣) كيف تختلف مفاهيم تحديد أحد المتغيرات المشيرة لآخر عن تحديد أحد متغيرات الإشارة لآخر ؟
- (٢٤) ماهي أنواع المقارنات التي يمكن إجراؤها بين المتغيرات المشيرة ؟ وماهي المؤثرات التي يمكن استخدامها في هذا الغرض ؟
- (٢٥) ماهي أنواع المقارنات التي يمكن إجراؤها بين متغيرات الإشارة ؟ هل يمكن أداء مثل هذه المقارنات على محتويات متغيرات ؟ وهل يمكن أدائها على حقول فردية ؟ ماهي القيود الموضوعة على مثل هذه المقارنات ؟
- (٢٦) كيف تتصل العناصر الفردية داخل قائمة متصلة طبقا لقاعدة LIFO مع بعضها ؟
- (٢٧) لخص المنطق المستخدم في إنتاج عناصر جديدة داخل قائمة متصلة طبقا لقاعدة LIFO . في أي ترتيب يتم إدخال العناصر ؟ وكيف تتصل العناصر مع بعضها ؟
- (٢٨) لخص المنطق المستخدم في عرض عناصر قائمة متصلة اتصال LIFO . في أي ترتيب تظهر العناصر ؟
- (٢٩) لخص المنطق المستخدم في إدخال عناصر داخل بعض المواقع المحددة في قائمة متصلة اتصال LIFO . كيف يعاد ضبط المشيريات ؟
- (٣٠) ماهو الغرض من عبارة dispose ؟ وماهو نوع المؤشرات المطلوب ؟ قارن ذلك مع عبارة new .
- (٣١) لخص المنطق المستخدم في حذف عنصر من قائمة متصلة اتصال LIFO . كيف يعاد ضبط المشيريات ؟

## Solved Problems

## مسائل محلولة :

- (٣٢) فيما يلي بعض تعريفات لأنواع مشيريات ، وأنواع متغيرات إشارة مناظرة لها .

```
(a)  TYPE next = {stockitem;
      stockitem = RECORD
        stockno : 1..9999;
        quantity : integer;
        nextitem : next
      END;
```

```

(b) TYPE line = PACKED ARRAY [1..25] OF char;
    pointer = ↑customer;
    customer = RECORD
        name : line;
        street : line;
        city : line;
        next : pointer
    END;

(c) TYPE next = ↑part;
    part = RECORD
        length : real;
        width : real;
        depth : real;
        nextpart : next
    END;

(d) TYPE values = ARRAY [1..3] OF real;
    pointer = ↑part;
    part = RECORD
        dimension : values;
        next : pointer
    END;

```

(٢٣) فيما يلي عدة تعريفات نوع وتوضيحات متغيرات مصاحبة لها ، تشمل مشيرات ومتغيرات الإشارة المناظرة لها .

```

(a) TYPE next = ↑stockitem;
    stockitem = RECORD
        stockno : 1..9999;
        quantity : integer;
        nextitem : next
    END;
    VAR firstitem, lastitem, thisitem : next;

    BEGIN
        .
        .
        new(thisitem);
        .
        .
    END.

(b) TYPE values = ARRAY [1..3] OF real;
    pointer = ↑part;
    part = RECORD
        dimension : values;
        next : pointer
    END;
    VAR workpart : pointer;

    BEGIN (* main action part *)
        .
        .
        new(workpart);
        .
        .
    END.

```

(٣٤) فيما يلي تخطيطات هيكلية لعدة برامج بسكال ، كل برنامج يحتوى على استخدام مشيريات ومتغيرات إشارة .

```
(a) PROGRAM sample(input,output);
    TYPE line = PACKED ARRAY [1..25] OF char;
    pointer = ↑customer;
    customer = RECORD
        name : line;
        street : line;
        city : line;
        next : pointer
    END;
    VAR nameandaddress : pointer;

    PROCEDURE readinput (VAR item : line);
    (* read a 25-character string *)
    BEGIN
        .
        .
        .
    END;

    BEGIN (* main action block *)
        new(nameandaddress);
        .
        .
        WITH nameandaddress↑ DO
            BEGIN
                readinput(name);
                readinput(street);
                readinput(city);
            END;
            .
            .
    END.
```

لاحظ استخدام مكون WHILE - DO لتسهيل قراءة عناصر متغير الإشارة ( انظر فصل ١٠ ) . لاحظ أيضا أن المؤشرات الفعلية المستخدمة مع readinput ، هي متغيرات سلسلة كما هو مطلوب في تعريف الإجراء .

```
(b) PROGRAM sample(input,output);
    TYPE line = PACKED ARRAY [1..25] OF char;
    pointer = ↑customer;
    customer = RECORD
        name : line;
        street : line;
        city : line;
        next : pointer
    END;
    VAR nameandaddress : pointer;

    PROCEDURE readinput (VAR personal : pointer);
    (* read the name, street and city *)
    BEGIN
        .
        .
        .
    END;
```



```

PROCEDURE writeoutput (personal : pointer);
(* display the name, street and city *)
BEGIN
.
.
.
END;

BEGIN (* main action block *)
  new(nameandaddress);
  .
  .
  readinput(nameandaddress);
  .
  .
  writeoutput(nameandaddress);
  .
  .
  dispose(nameandaddress);
  .
  .
END.

```

لاحظ أن المؤشرات الفعلية هي متغيرات من نوع المشير الآن : ( قارن ذلك مع المثال السابق ) لاحظ أيضا أن readinput يستخدم مؤشراً متغيراً ، بينما يستخدم writeoutput مؤشر قيمة (وكل منهما مؤشر من نوع المشير) .

```

(c) PROGRAM sample(input,output);
TYPE values = ARRAY [1..3] OF real;
   pointer = ↑part;
   part = RECORD
       dimension : values;
       next : pointer
   END;
VAR firstpart,nextpart : pointer;
    area,volume : real;

BEGIN (* main action statements *)
  new(nextpart);
  .
  .
  .
  nextpart↑.dimension[1] := 1.0;
  nextpart↑.dimension[2] := 2.5;
  nextpart↑.dimension[3] := 0.3;
  .
  .
  .
  nextpart↑.next := firstpart;
  .
  .
  WITH nextpart↑ DO
    BEGIN
      area := dimension[1] * dimension[2];
      volume := area * dimension[3];
      IF dimension[1] <> dimension[3] THEN . . .
        ELSE . . .

```

```

END;
.
.
IF nextpart↑.next = NIL THEN . . .
ELSE . . . ;
.
.
dispose(nextpart);
.
.
END.

```

العبارات الموجودة في هذا المثال ليست مرتبطة مع بعضها ارتباطاً منطقياً ، فهي موجهة ببساطة لتوضيح تحديدات ومقارنات مختلفة لمتغيرات مشيرة ، وعناصر متغيرات إشارة .

### مشاكل متكاملة : Supplementary Problems

(٣٥) اكتب أنواع تعريفات مناسبة ، أو توضيحات متغيرات لكل موقف من المواقف المذكورة أدناه :

١ - عرف نوع مشير اسمه link يسمى متغير الإشارة المناظر له customer ، ويحتوى على الخطوات التالية :

١ - name ( ٢٠ خانة )

٢ - acctno ( عدد صحيح يتراوح من ١ إلى ٩٩٩٩ )

٣ - balance ( كمية حقيقية ) .

٤ - next ( مشير يشير إلى السجل التالي ) .

ب - صف متغيرين من نوع المشير ، كل منهما من نفس نوع link . أعط اسمي nextcustomer , lastcustomer لهما .

ج - انتج ، ثم بعد ذلك احذف متغيراً ديناميكياً يناظر nextcustomer . لاحظ أن nextcustomer هو متغير من النوع المشير .

(٣٦) تشير المشاكل التالية إلى متغيرات من النوع المشير ، وإلى متغيرات الإشارة المناظرة لها ، والمعرفة في المشكلة السابقة .

١ - حدد قيمة المشير في nextcustomer↑ للمتغير المشير lastcustomer .

ب - هل تشير قيمة المؤشر في nextcustomer↑ إلى لاشئ ؟

ج - حدد كل عنصر من عناصر nextcustomer↑ للعنصر الذي يناظره في lastcustomer↑ .

د - حدد الموازنة \$ 287.55 لـ nextcustomer الذي يكون رقم حسابه 1330 .

هـ - احصل على قيمة من قيم  $\uparrow \text{nextcustomer.name}$  ، وذلك عن طريق الاتصال بإجراء يسمى  $\text{readinput}$  .

( أدخل تخطيطاً لـ  $\text{readinput}$  في حلك ) .

و - خطط مكون WHILE - DO يستمر في التنفيذ ، طالما أن قيمة  $\text{nextcustomer}$  ليست NIL .

(٣٧) فيما يلي عدة تخطيطات هيكلية لبرامج تحتوي على استخدام مشيرات ، ومتغيرات إشارة مناظرة لها . وبعض العبارات مكتوبة خطأ ، عرف كل الأخطاء .

```
(a) PROGRAM sample(input,output);
    TYPE textline = PACKED ARRAY [1..80] OF char;
    link = ↑pointer;
    stockitem = RECORD
        stockno : 1..9999;
        stocktype : textline;
        next : pointer
    END;
    VAR newitem,olditem : stockitem;
    BEGIN
        .
        .
        .
    END.
```

```
(b) PROGRAM sample(input,output);
    TYPE link = ↑player;
    player = RECORD
        ssn : PACKED ARRAY [1..9] OF char;
        position : PACKED ARRAY [1..12] OF char;
        age : 1..99;
        height : real;
        weight : real
    END;
    VAR nextplayer,lastplayer : link;
    BEGIN
        .
        new(player);
        .
        .
        IF player↑.age < 40 THEN . . . ;
        .
        .
        IF nextplayer <> NIL THEN . . . ;
        .
        .
    END.
```

```
(c) PROGRAM sample(input,output);
    TYPE textline = PACKED ARRAY [1..30] OF char;
    link = ↑student;
    student = RECORD
        name : textline;
        next : link
    END;
    VAR point,tag : link;
```

```

PROCEDURE readtext (VAR line : textline);
VAR count : 1..30;
BEGIN
  FOR count := 1 TO 30 DO line[count] := ' ';
  count := 1;
  REPEAT
    read(line[count]);
    count := succ(count)
  UNTIL eoln;
  readln
END;

BEGIN (* main action block *)
  new(point);
  readtext(point^.name);
  point^.next := NIL;
  tag := point;
  REPEAT
    new(point);
    readtext(point^.name);
    point^.next := tag;
    tag := point
  UNTIL point^.name[1] = '*';
  tag := point^.next
END.

```

## Programming Problems

## مشاكل برمجة

(٣٨) عدل البرنامج المعطى فى مثال (١٣ - ٨) ، بحيث إن كل عنصر داخل القائمة ( أى كل سجل ) يحتوى على اسم وعنوان الشارع والمدينة والولاية ورقم البريد . ضع الاسم على سطر خاص به ، وعنوان الشارع على سطر آخر ، وبقية المعلومات على سطر ثالث .

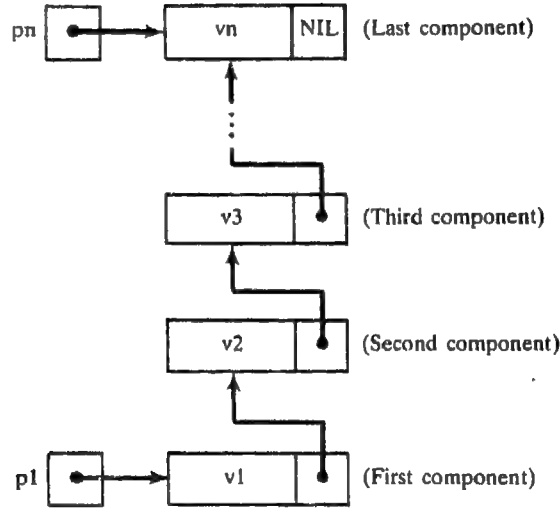
دع السجل يحتوى حقلين أول حقل ، عبارة عن منظومة ذات بعدين ، بها ٣ صفوف ، و ٨٠ عموداً ، ويمثل كل صف سطرًا من أسطر النص . والحقل الثانى عبارة عن مؤشر إلى السجل التالى .

(٣٩) عدل البرنامج المعطى فى مثال (١٣ - ٩) ، بحيث يمكن تطبيقه على كل من التكوينات المتصلة التالية :

أ - قائمة متصلة خطية ، بها فئتان من المشيرات ( فئة تشير فى الاتجاه الامامى ، والفئة الأخرى تشير فى الاتجاه الخلفى )

ب - قائمة متصلة دائرية ( استخدم متغيراً مشيراً لتعريف بداية القائمة )

ج - قائمة متصلة خطية اتصال LIFO كما هو موضح فى شكل ١٣ - ٨



شكل ١٣-٨

د - شجرة ثنائية بعدد مستويات محدد ( لاحظ أن الطريقة المنطقية لحركة الشجرة مطلوبة ) .

(٤٠) اكتب برنامجاً كاملاً بلغة البسكال ، يسمح لك بإدخال شجرة العائلة الخاصة بك المعدة بالكمبيوتر . ابدأ بتحديد عدد الأجيال ( أى عدد المستويات ) ، ثم أدخل الاسماء والجنسيات لتعديل الشجرة ، وإضافة أسماء جديدة ( انظر النقاط ) فى الشجرة . أدخل أيضاً جزءاً لعرض من محتويات الشجرة تلقائياً بعد كل تجديد .

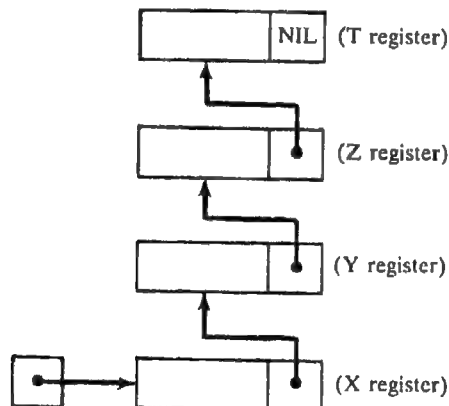
اختبر البرنامج ، مبتدئاً بأول زوج من أجداد والديك ، وزوجاً من أجداد أجداد والديك .

(٤١) تستخدم إحدى الحاسبات RPN مخططاً ، بحيث إن كل قيمة عددية جديدة تتبعها عملية تنفيذ على القيمة الجديدة والقيمة التى سبقتها . وحروف RPN هى اختصار Reverse Polish Notation التمييز البولندي للحركة . وعلى هذا ... فإضافة عددين ، وليكونا 3.3 , 4.8 ، يتطلب الضغط على المفاتيح التالية :

3.3    (enter)  
4.8    +

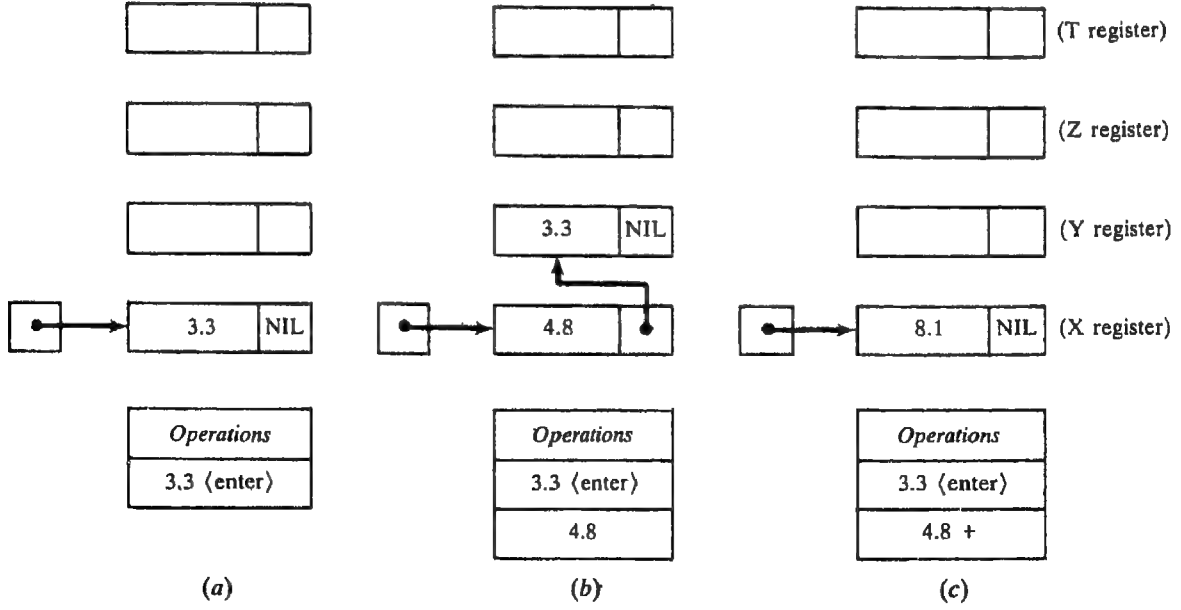
والمجموع 8.1 يعرض فى الحاسبة

تستخدم الحاسبات من هذا النوع حزمة تحتوى عادة على أربعة مسجلات ( أربعة عناصر ) كما هو موضح فى شكل ١٣-٩ . كل عدد جديد يتم إدخاله فى السجل X متسبباً فى دفع كل القيم التى سبق إدخالها لأعلى فى الحزمة . فإذا ما كان المسجل T مشغولاً من قبل ، فسوف يفقد العدد القديم ( حيث إن القيمة التى تدفع إليه من المسجل Z تحل محله ) .



شكل ١٣-٩

وتنفذ العمليات الحسابية دائما على الأعداد الموجودة في المسجل X ، والمسجل Y . وتظهر نتيجة مثل هذه العمليات دائما في المسجل X ، متسببه في أن كل شيء في المسجلات العلوية يسقط مستويا واحدا لأسفل ( أى تحرك الحزمة ) . هذه العملية موضحة في الأشكال (a) 13 - 10 ، (b) 13 - 10 ، (c) 13 - 10 بالنسبة لإجراء عملية جمع عددين سالفة الذكر .



شكل ١٣ - ١٠

اكتب برنامجا متاخلا بلغة البسكال ، يحاكي الحاسبة من نوع RPN . اعرض محتويات الحزمة بعد كل عملية ، كما في الأشكال (a) 13 - 10 ، (b) 13 - 10 ، (c) 13 - 10 ، على أن يكون البرنامج جزءاً لأداء كل من العمليات التالية :

المفاتيح	العملية
( enter ) ( القيمة )	إدخال بيانات جديدة
+	الجمع
-	الطرح
*	الضرب
/	القسمة

اختبر البرنامج مستخدماً أى بيانات عديدة من اختيارك .

## ملحق (أ)

## الكلمات المحجوزة

AND	END	NIL	SET
ARRAY	FILE	NOT	THEN
BEGIN	FOR	OF	TO
CASE	FUNCTION	OR	TYPE
CONST	GOTO	PACKED	UNTIL
DIV	IF	PROCEDURE	VAR
DO	IN	PROGRAM	WHILE
DOWNT0	LABEL	RECORD	WITH
ELSE	MOD	REPEAT	

## ملحق (ب)

## المعرفات القياسية

abs	false	pack	sin
arctan	get	page	sqr
boolean	input	pred	sqrt
char	integer	put	succ
chr	ln	read	text
cos	maxint	readln	true
dispose	new	real	trunc
eof	odd	reset	unpack
eoln	ord	rewrite	write
exp	output	round	writeln

## ملحق (ج)

## الإجراءات القياسية

الإجراء	الغرض منه	الإشارة إليه
dispose	حذف متغير ديناميكي ، أى حذف عنصر من عناصر قائمة متصلة	انظر القسم ٥ - ١٣
get	نقل عناصر بيانات من ملف مدخلات إلى احتياطي ملف	انظر القسم ٥ - ١١
new	إنتاج متغير ديناميكي ، أى إضافة عنصر إلى قائمة متصلة	انظر القسم ٣ - ١٣
pack	ينتج عنه ضغط عناصر البيانات في المخزن ، أى داخل ذاكرة الكمبيوتر	انظر القسم ٤ - ٩
page	إنتاج مخرجات جديدة ، تبدأ في بداية صفحة جديدة	انظر القسم ٧ - ٤
put	نقل عناصر بيانات من احتياطي الملف إلى ملف مخرجات	انظر القسم ٣ - ١١
read	قراءة عناصر بيانات من ملف مدخلات	انظر القسم ٢ - ٤ والقسم ٦ - ١١
readln	قراءة عناصر بيانات من ملف مدخلات ، ثم النقل للسطر التالي	انظر القسم ٣ - ٤
reset	إعداد الملف للقراءة	انظر القسم ٥ - ١١
rewrite	إعداد الملف للكتابة	انظر القسم ٣ - ١١
unpack	فك ضغط البيانات لتشغيل حيز أكبر من ذاكرة الكمبيوتر	انظر القسم ٤ - ٩
write	كتابة بيانات في ملف مخرجات	انظر القسم ٥ - ٤ والقسم ٤ - ١١
writeln	كتابة عناصر بيانات في ملف مخرجات ، ثم النقل للسطر التالي .	انظر القسم ٦ - ٤



## ملحق (د)

## الدوال القياسية

الدالة	الفرض منها	نوع المؤشر	نوع النتيجة
abs (x)	حساب القيمة المطلقة	صحيح أو حقيقي	مثل x
arctan (x)	حساب الظل العكسي	صحيح أو حقيقي	حقيقي
chr (x)	تحديد الرمز الذي يمثله	صحيح أو حقيقي	حرفي
cos (x)	حساب جيب تمام x ( x بالتقدير الدائري )	صحيح أو حقيقي	حقيقي
eof (x)	تحديد ما إذا انتهى الملف أم لا	ملف	بوليان
eoln (x)	تحديد ما إذا انتهى السطر أم لا	ملف	بوليان
exp (x)	حساب $e^x$ حيث $e = 2.7182818$ ، وهي أساس النظام الطبيعي للوغاريتمات	صحيح أو حقيقي	حقيقي
ln (x)	حساب اللوغاريتم الطبيعي لـ x ( $x > 0$ )	صحيح أو حقيقي	حقيقي
odd (x)	تحديد ما إذا كانت x فردية أم زوجية ( مع إعادة قيمة صحيح ( true ) إذا كانت x فردية ، وقيمة خاطئ false إذا كانت x زوجية )	صحيح	بوليان
ord (x)	تحديد العدد العشري الصحيح المستخدم في عمل شفرة الحرف x .	حرفي	صحيح
pred (x)	تحديد ما يسبق x .	صحيح أو حرفي أو بوليان	مثل نوع x
round (x)	تقريب قيمة x لأقرب رقم صحيح	حقيقي	صحيح
sin (x)	حساب جيب الزاوية x ( x بالتقدير الدائري )	صحيح أو حقيقي	حقيقي
sqr (x)	حساب مربع x .	صحيح أو حقيقي	مثل نوع x
sqrt (x)	حساب الجذر التربيعي لـ x ( $x \geq 0$ )	صحيح أو حقيقي	حقيقي
succ (x)	تحديد ما يلي x	صحيح أو حرفي أو بوليان	مثل نوع x
trunc (x)	حذف الكسر العشري من x	حقيقي	صحيح

## ملاحق ( هـ )

المؤثرات  
Operators

مؤثرات البسكال ملخصة أدناه ، وذلك طبقا لأولويات التنفيذ الطبيعية ( الأعلى فالأقل )

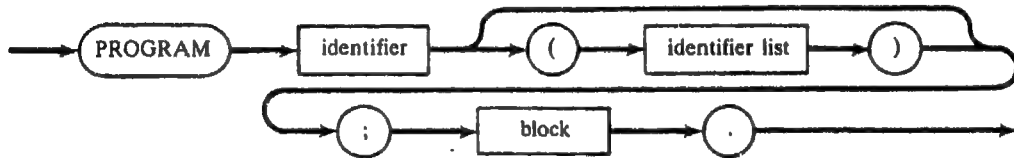
المؤثر ( المؤثرات )	أولوية التنفيذ
NOT	1 ( الأعلى )
* / DIV MOD AND	2
+ - OR	3
= < > <= > >= IN	4 ( الأقل )

المؤثرات الموجودة فى نفس المستوى تنفذ كلها من اليمين إلى اليسار ، بغض النظر عن نوع المؤثر نفسه .

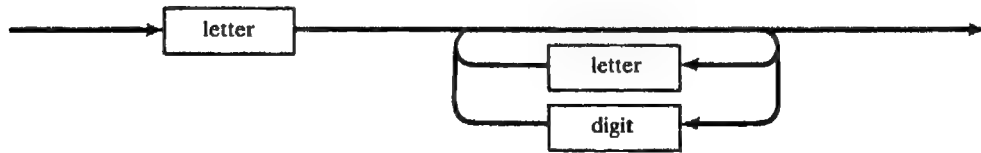
## ملحق (و)

رسومات التكوينات

## Syntax Diagrams



شكل (و-١) : برنامج



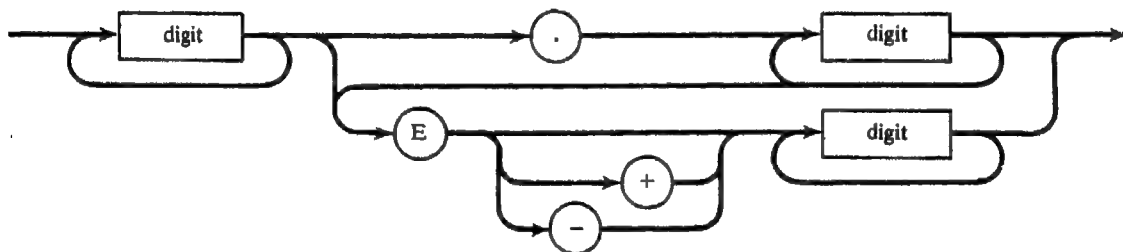
شكل (و-٢) : معرف



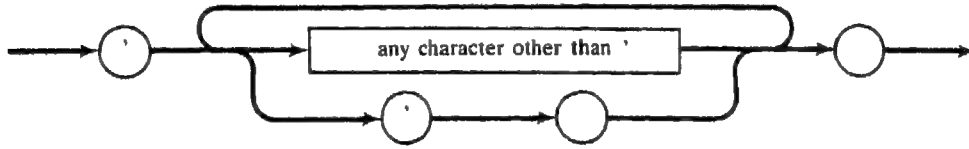
شكل (و-٣) : قائمة معرفات



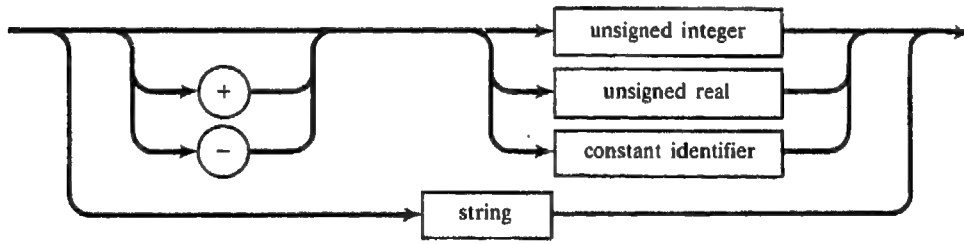
شكل (و-٤) : صحيح بدون إشارة



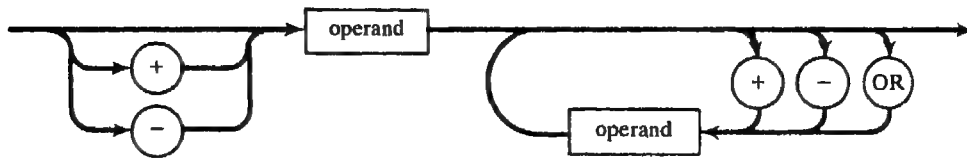
شكل (و-٥) : حقيقي بدون إشارة



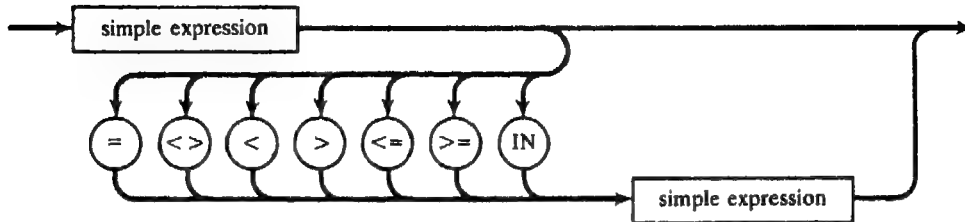
شكل (و-٦) : سلسلة



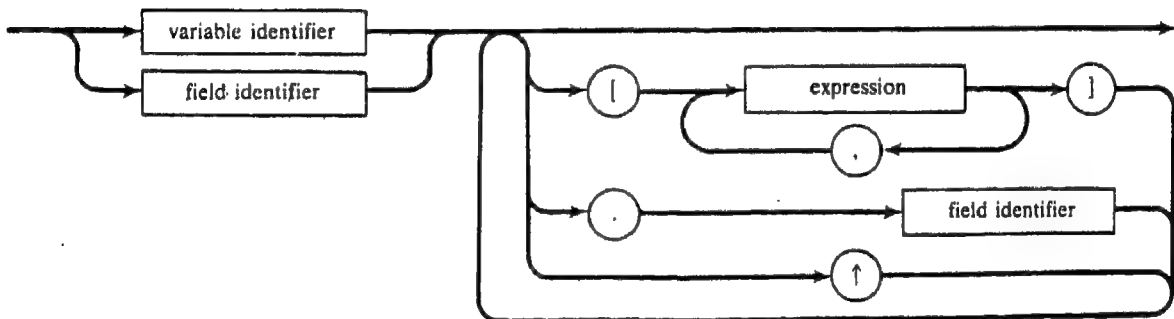
شكل (و-٧) : ثابت



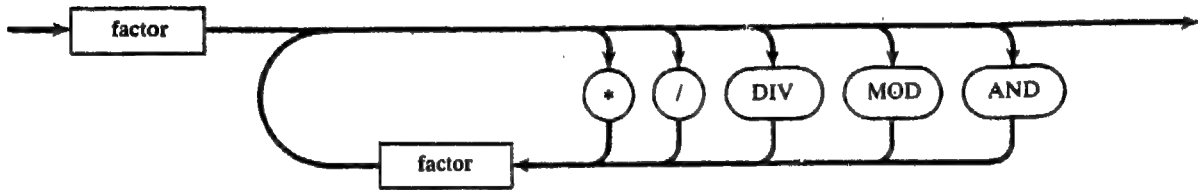
شكل (و-٨) : تعبير بسيط



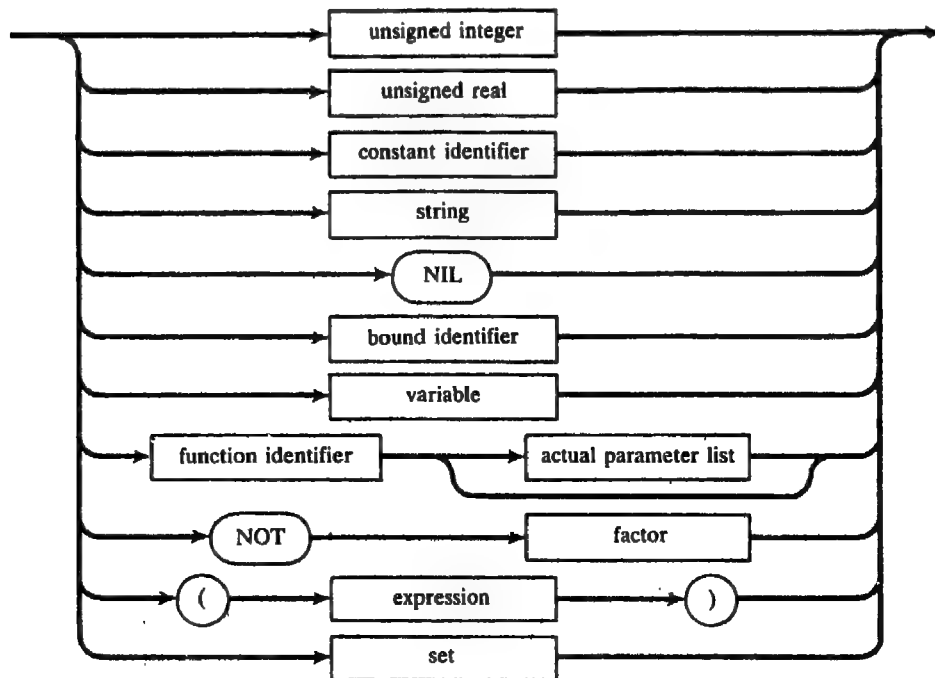
شكل (و-٩) : تعبير



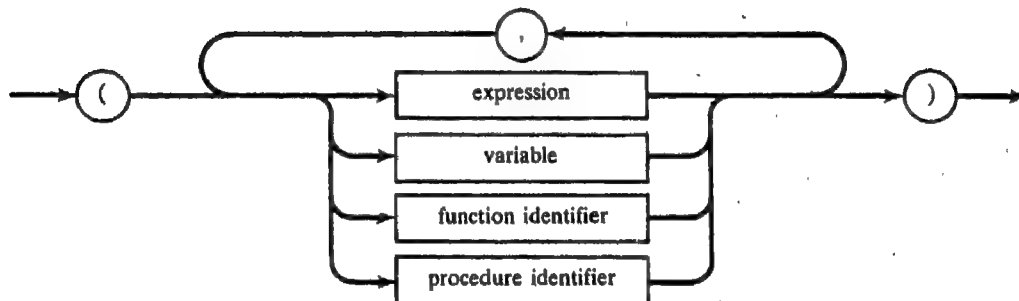
شكل (و-١٠) : متغير



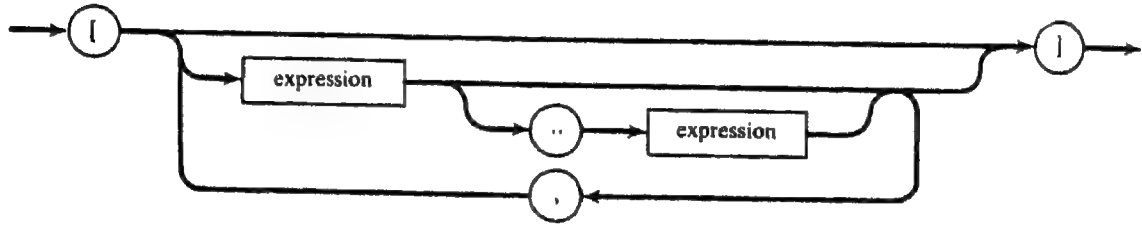
شكل (و- ١١) : عنصر يجرى عليه عملية



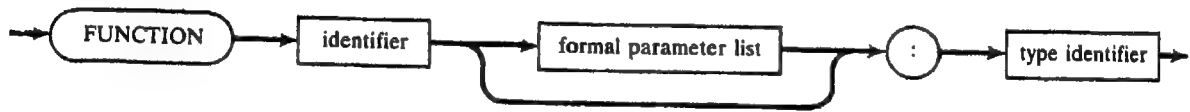
شكل (و- ١٢) : معامل



شكل (و- ١٣) : قائمة مؤشرات فعلية



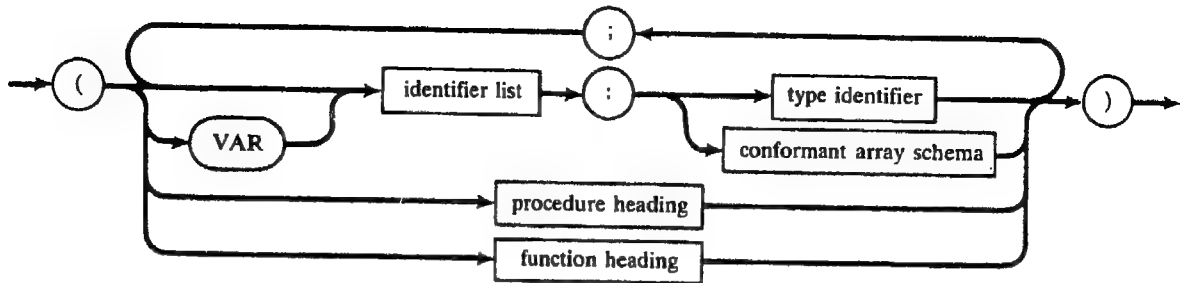
شكل (و- ١٤) : فئة



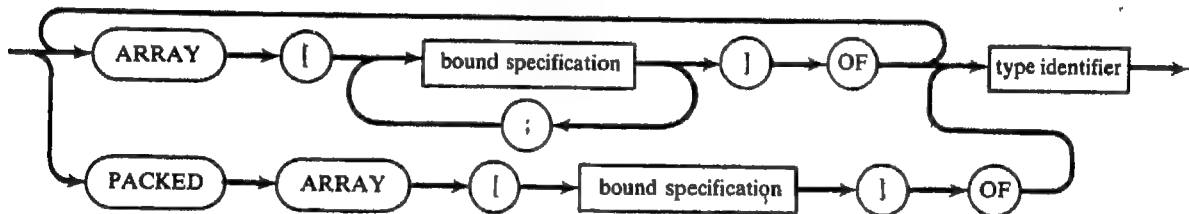
شكل (و- ١٥) : عنوان دالة



شكل (و- ١٦) : عنوان إجراء

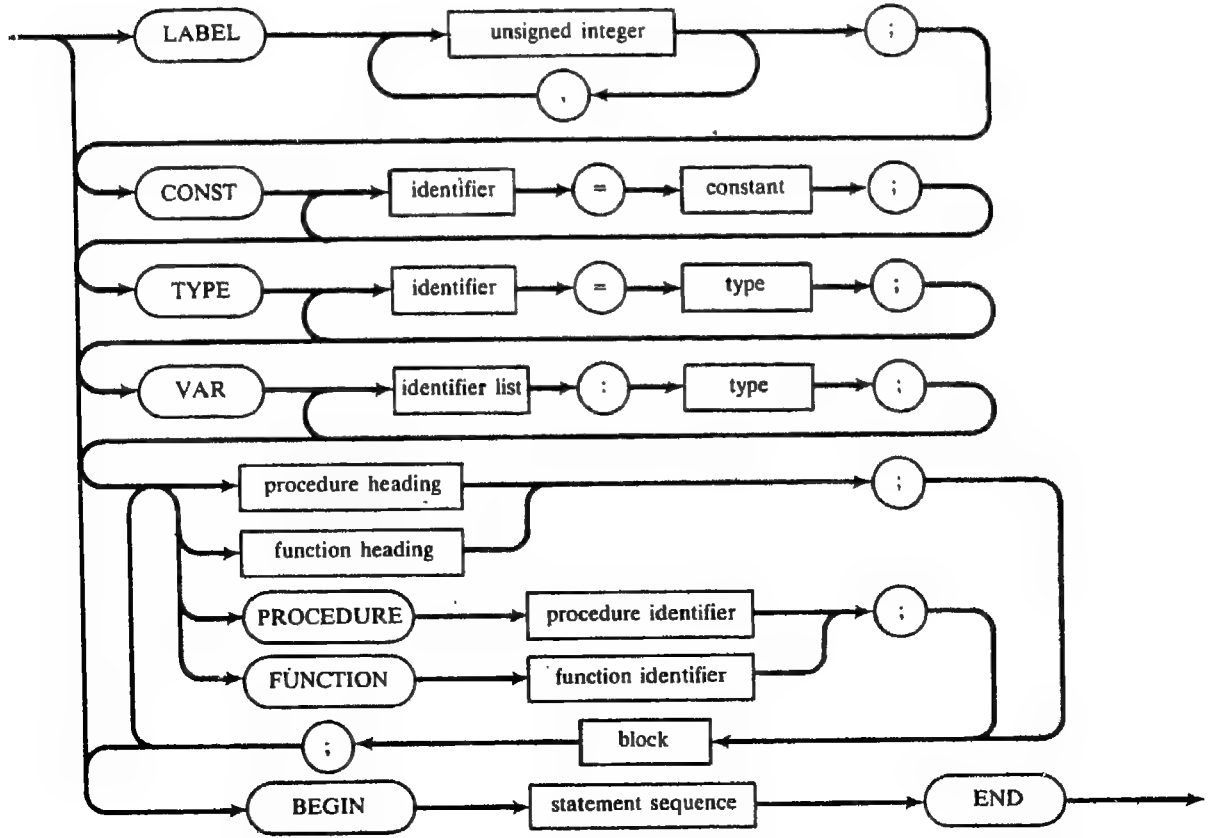


شكل (و- ١٧) : قائمة مؤشرات رسمية

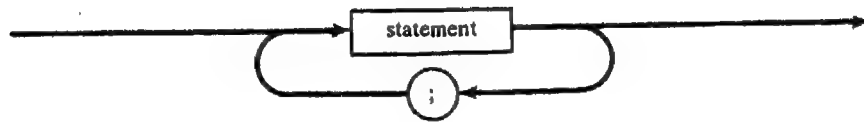


شكل (و- ١٨) : مخطط منظومة



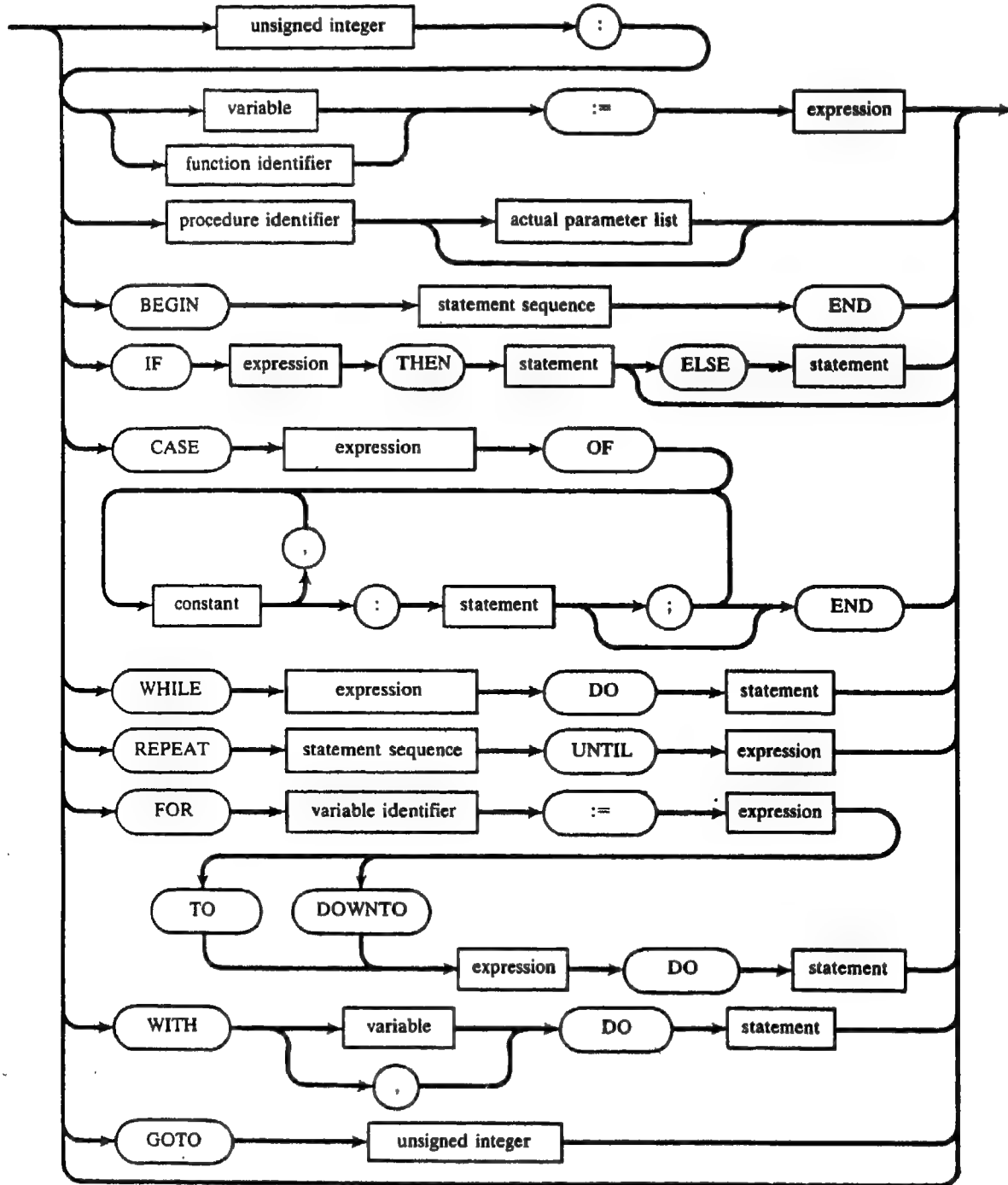


شكل (و- ٢٢) : المجموعة



شكل (و- ٢٣) : تسلسل العبارة





شكل (و- ٢٤) : العبارة

## ملحق (ز)

## فئة رموز ASCII

ASCII Value	Character	ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
000	NUL	032	blank	064	@	096	`
001	SOH	033	!	065	A	097	a
002	STX	034	"	066	B	098	b
003	ETX	035	#	067	C	099	c
004	EOT	036	\$	068	D	100	d
005	ENQ	037	%	069	E	101	e
006	ACK	038	&	070	F	102	f
007	BEL	039	'	071	G	103	g
008	BS	040	(	072	H	104	h
009	HT	041	)	073	I	105	i
010	LF	042	*	074	J	106	j
011	VT	043	+	075	K	107	k
012	FF	044	,	076	L	108	l
013	CR	045	-	077	M	109	m
014	SO	046	.	078	N	110	n
015	SI	047	/	079	O	111	o
016	DLE	048	0	080	P	112	p
017	DC1	049	1	081	Q	113	q
018	DC2	050	2	082	R	114	r
019	DC3	051	3	083	S	115	s
020	DC4	052	4	084	T	116	t
021	NAK	053	5	085	U	117	u
022	SYN	054	6	086	V	118	v
023	ETB	055	7	087	W	119	w
024	CAN	056	8	088	X	120	x
025	EM	057	9	089	Y	121	y
026	SUB	058	:	090	Z	122	z
027	ESC	059	;	091	[	123	{
028	FS	060	<	092	\	124	
029	GS	061	=	093	]	125	}
030	RS	062	>	094	↑	126	~
031	US	063	?	095	—	127	DEL

ملاحظة: أول ٣٢ رمز وآخر رمز ، هي رموز تحكم لا يمكن طباعتها

## إجابات المشاكل المتكاملة

### الفصل الثانى

- ٥٠ - (a) - صحيح .  
 (b) - صحيح .  
 (c) - file هي كلمة محجوزة .  
 (d) - صحيح .  
 (e) - لايسمح بوجود فراغات .  
 (f) - لايسمح باستخدام رموز غير الحروف والأرقام .  
 (g) - يجب أن يكون أول رمز عبارة عن حرف أبجدى ، كما أن الشرطة غير مسموح بها أيضا .

- ٥١ - (a) - صحيح ( من النوع الحقيقى ) .  
 (b) - لايسمح بوجود الفواصل .  
 (c) - صحيح ( من النوع الحقيقى )  
 (d) - قد كون قيمة الاس كبيرة جدا .  
 (e) - يجب أن يظهر رقم صحيح قبل العلامة العشرية وبعدها .  
 (f) - رمز غير مسموح به .  
 (g) - صحيح ( من النوع الحقيقى ) .  
 (h) - صحيح ( من النوع الحقيقى ) قد يكون كبيرا بالنسبة لبعض أجهزة الكمبيوتر .  
 (i) - صحيح ( من النوع الحقيقى ) .

- ٥٢ - (a) - صحيح .  
 (b) - لايمكن استخدام علامتى التنصيص حول السلسلة .  
 (c) - الفاصلة فى الناحية اليمنى غير موجودة .  
 (d) - صحيح .  
 (e) - صحيح .  
 (f) - صحيح .

٥٣  
 CONST month = 'july';  
 fica = '123-45-6789';  
 price = '\$95.00'; (or price = 95.00;)  
 gross = 2500.00;  
 partno = 48837;  
 bound = 0.00391;

٥٤  
 VAR period,status : char;  
 terminal : boolean;  
 index,row : integer;  
 clearance : real;

- ٥٥ - (a) - هذه عبارة تحديد ، وليست تعبير .  
 (b) - عددي أو بولياني ، وذلك طبقا لنوع البيانات المصاحبة للقيمة .  
 (c) - عددي .  
 (d) - بولياني .  
 (e) - غير مسموح بوجود مؤثرين متتاليين .  
 (f) - عددي .  
 (g) - بولياني .
- ٥٦ - (a) - بسيط ( عبارة تحديد )  
 (b) - مرتب ( مركب ) .  
 (c) - مرتب ( تكراري )  
 (d) - مرتب ( شرطي )  
 (e) - بسيط ( تحديد )  
 (f) - بسيط ( إجرائي )  
 (g) - بسيط ( نقل غير شرطي )  
 (h) - بسيط ( تحديد )
- ٥٧ يتم الاتصال بالإجراءات في المشاكل 2.56(c) , 2.56(f) . ويتم الاتصال بالدالة في المشكلة رقم 2.56(e) .
- ٥٨ - (a) - يبدأ برنامج البسكال بكلمة البسكال المحجوزة ، يتبعها معرف ، ثم قائمة اختيارية بمعرفات موضوعات بين قوسين ، ثم تلي ذلك فاصلة منقوطة . وتلي هذه العناصر مجموعة البرنامج ، أو صلب البرنامج .  
 (b) - يوضح هذا الرسم طرقا عديدة مختلفة لكتابة عدد حقيقي لإشارة واحدة له . وهي مايلي :  
 ١ - رقم صحيح واحد أو أكثر ، يتبعه علامة عشرية ، وبعد ذلك رقم صحيح واحد أو أكثر .  
 ٢ - رقم صحيح واحد أو أكثر ، يتبعه الحرف E ، وبعد ذلك رقم صحيح واحد أو أكثر ، وتظهر إشارة بين الحرف E والأرقام التي تليه ( هذه هي الصيغة الأسية ) .  
 ٣ - خليط مما سبق ، أي رقم صحيح واحد أو أكثر ، تليه علامة عشرية ، ثم رقم صحيح ، تليه هذه العناصر ، والحرف يلي ذلك إشارة موجب أو إشارة سالب اختيارية ، ثم رقم صحيح واحد أو أكثر .  
 (c) - يبين هذا الشكل طرقا عديدة مختلفة لكتابة سلسلة ، وهي مايلي :  
 ١ - رقم صحيح بدون إشارة تسبقه إشارة موجب أو إشارة سالب اختيارية .  
 ٢ - رقم حقيقي بدون إشارة تسبقه إشارة موجب أو إشارة سالب اختيارية .  
 ٣ - معرف ثابت تسبقه إشارة موجب أو إشارة سالب اختيارية .  
 ٤ - سلسلة .

## الفصل الثالث

- ٤٠ - (a) - حقيقي 0.3333333 - .  
 (b) - صحيح 2 - .  
 (c) - حقيقي 0.3333333 - .  
 (d) - صحيح 14 - .  
 (e) - حقيقي 0.03 - .  
 (f) - صحيح 3 - .

- ٤١ . (a) - خطأ .  
 . (b) - صحيح .  
 . (c) - صحيح .  
 . (d) - صحيح .  
 . (e) - صحيح .  
 . (f) خطأ .  
 . (g) خطأ .  
 . (h) - صحيح .
- ٤٢ (a) 4.667 (i) True  
 (b) C (j) False  
 (c) 101 (k) True  
 (d) 9 (l) 2  
 (e) d (m) 3  
 (f) False (n) -2  
 (g) 11 (o) -3  
 (h) f
- ٤٣ (b) الرمز المناظر للشفرة 67 في نظام EBCDIC غير موجود في جدول 3.2 .  
 133 (C)  
 (t) الرمز المناظر للشفرة 252 في نظام EBCDIC غير موجود في جدول 3.2 .  
 192 (u)
- ٤٤ (a) يتطلب DIV عنصراً صحيحاً .  
 (b) - صحيح .  
 (c) لا يمكن تنفيذ العمليات الحسابية على عناصر سلسلة .  
 (d) صحيح .  
 (e) - لا يوجد مؤثر في المقام .  
 (f) صحيح .  
 (g) يجب أن توضع عناصر بوليان بين قوسين .  
 (h) - صحيح .  
 (i) - المؤثرات غير متوافقة .  
 (j) - الأقواس غير متوازنة .  
 (k) - ينتج عن هذا التعبير قيمة تتعدى maxint .  
 (l) - غير مسموح بمؤثرات متتالية .  
 (m) دالة Odd تتطلب مؤشراً صحيحاً .  
 (n) - صحيح .

```
VAR gross,net,tax : real;
    employee : integer;
    status,sex : char;
    exempt : boolean;
```

### الفصل الرابع

٤٠. يتم إدخال بيانات المدخلات في سطرين ، على أن يوجد ثلاث كميات صحيحة ، وكميتان حقيقيتان في أول سطر ، ويوجد أربعة رموز متتالية في السطر الثاني .  
وتحتوى المخرجات على ثلاثة أسطر ، بينها مسافات مزبوجة ، يحتوى أول سطر على قيم أولى ثابتين flag , factor . أما السطر الثاني ، فيحتوى على القيم الحالية للثلاثة متغيرات الصحيحة والمتغيرين الحقيقيين .  
أما السطر الثالث ، فيحتوى على القيم الحالية للأربعة متغيرات من النوع الحرفى ومتغيرى بوليان .  
وكل عناصر المخرجات الفردية تفصل عن بعضها ، فيما عدا الأربعة قيم المتتالية من النوع الحرفى الموجودة في السطر الثاني .

٤١

```
(a) i1=1    i2=2    i3=3    r1=4.0    r2=5.0
    c1=b    c2=l    c3=u    c4=e
(b) i1=1    i2=2    i3=3    r1=4.0    r2=5.0
    c1=b    c2=(blank) c3=(blank) c4=(blank)
(c) i1=1    i2=2    i3=3    r1=4.0    r2=5.0
    c1, c2, c3 and c4 will be undefined
(d) i1=1    i2=2    i3=3    r1=4.0    r2=5.0
    c1=g    c2=r    c3=e    c4=e
```

٤٢

```
red 0.5000000E-02
(blank line)
    100    -200    -300 0.4004440E+03-0.5005550E+03
(blank line)
PINK true false
```

٤٣

```
flag=red factor= 0.005
(blank line)
i1= 100 i2=-200 i3=-300 r1= 400.4 r2=-500.6
(blank line)
color=PINK b1= true b2=false
```

## تابع الفصل الرابع

```

-----
flag=red factor= 0.005      (top of page)
(blank line)
i1= 100 i2= -200 i3= -350
r1= 400.44 r2= -500.56
-----
color=PINK                  (top of page)
(blank line)
b1= true b2= false

```

٤٤

- (a) `readln(i1,i2,i3,r1,r2,c1,c2,c3,c4);` ٤٥
- (b) `readln(i1,r2,c3,c4);`
- (c) `readln(i1,i2,i3);`  
`readln(r1,r2);`  
`readln(c1,c2,c3,c4);`
- (d) `readln(i1);`  
`readln(i2);`  
`readln(i3);`  
`readln(r1);`  
`readln(r2);`  
`readln(c1);`  
`readln(c2);`  
`readln(c3);`  
`readln(c4);`
- (e) `write(flag,factor:6:3,i1:5,i2:5,i3:5,r1:9:3,r2:9:3);`  
`writeln(' ',c1,' ',c2,' ',c3,' ',c4,b1,b2);`  
 (Other solutions are also possible.)
- (f) `write(flag,factor:8:2,i1:4,i2:4,i3:4,r1:8:2,r2:8:2);`  
`writeln(' ',c1,' ',c2,' ',c3,' ',c4,b1:6,b2:6);`
- (g) `writeln(flag,factor:6);`  
`writeln;`  
`writeln(c1,' ',c2,' ',c3,' ',c4,b1:6,b2:6);`  
`writeln;`  
`writeln(i1:5,i2:5,i3:5,r1:9:3,r2:9:3);`
- (h) `writeln('flag=',flag,' factor=',factor);`  
`writeln;`  
`writeln('color=',c1,c2,c3,c4,' b1=',b1,' b2=',b2);`  
`writeln;`  
`write('i1=',i1:4,' i2=',i2:4,' i3=',i3:4);`  
`writeln(' r1=',r1:8:3,' r2=',r2:8:3);`
- (i) `page;`  
`writeln(flag,i1:5,r1:9:3,' ',c1,' ',c2,b1:6);`  
`writeln;`  
`writeln;`  
`writeln(factor,i2:5,i3:5,r2:9:3,' ',c3,' ',c4,b2:6);`  
 or  
`writeln('1',flag,i1:5,r1:9:3,' ',c1,' ',c2,b1:6);`  
`writeln;`  
`writeln('0',factor,i2:5,i3:5,r2:9:3,' ',c3,' ',c4,b2:6);`
- (j) `page;`  
`writeln(flag,i1:5,r1:7:2,' ',c1,' ',c2,b1:7);`  
`writeln;`  
`writeln;`  
`writeln(factor:7:2,i2:5,i3:5,r2:7:2,' ',c3,' ',c4,b2:7);`

```
(k) page;
writeln('flag=',flag);
writeln('factor=',factor:12);
writeln('i1=',i1:4);
writeln('i2=',i2:4);
writeln('i3=',i3:4);
writeln('r1=',r1:12);
writeln('r2=',r2:12);
writeln('c1=',c1);
writeln('c2=',c2);
writeln('c3=',c3);
writeln('c4=',c4);
writeln('b1=',b1:5);
writeln('b2=',b2:5);
```

SUM= 60 PRODUCT= 6000

٤٦

(Note that there is one blank space before 60, two before 6000.)

## – الفصل الخامس

```
(a) PROGRAM hello(output);
(* print HELLO! at the beginning of a line *)
BEGIN
    writeln('HELLO!')
END.

(b) PROGRAM friends(input,output);
(* interactive HELLO program *)
VAR c1,c2,c3,c4,c5,c6 : char;
BEGIN
    write('HI, WHAT'S YOUR NAME? ');
    readln(c1,c2,c3,c4,c5,c6);
    writeln;
    writeln;
    writeln('WELCOME ',c1,c2,c3,c4,c5,c6);
    writeln('LET'S BE FRIENDS!')
END.

(c) Noninteractive version:
PROGRAM celsius(input,output);
(* convert temperature from Fahrenheit to Celsius *)
VAR f,c : real;
BEGIN
    readln(f);
    c := (5/9)*(f-32);
    writeln('F=',f:7:2, ' C=',c:7:2)
END.

Interactive version:
PROGRAM celsius2(input,output);
(* convert temperature from Fahrenheit to Celsius *)
(* interactive version *)
VAR f,c : real;
BEGIN
    write('F= ');
    readln(f);
    c := (5/9)*(f-32);
    writeln('C=',c:7:2)
END.
```



## (d) Noninteractive version:

```

PROGRAM piggybank(input,output);
(* piggy-bank problem *)
VAR n1,n2,n3,n4,n5 : integer;
    dollars : real;
BEGIN
    readln(n1,n2,n3,n4,n5);
    dollars := 0.5*n1 + 0.25*n2 + 0.1*n3 + 0.05*n4 + 0.01*n5;
    writeln;
    writeln('Half-dollars: ',n1:3);
    writeln('Quarters: ',n2:3);
    writeln('Dimes: ',n3:3);
    writeln('Nickels: ',n4:3);
    writeln('Pennies: ',n5:3);
    writeln;
    writeln('The piggy bank contains ',dollars:6:2,' dollars')
END.

```

## Interactive version:

```

PROGRAM piggybank(input,output);
(* interactive piggy-bank problem *)
VAR n1,n2,n3,n4,n5 : integer;
    dollars : real;
BEGIN
    write('How many half-dollars? ');
    readln(n1);
    write('How many quarters? ');
    readln(n2);
    write('How many dimes? ');
    readln(n3);
    write('How many nickels? ');
    readln(n4);
    write('How many pennies? ');
    readln(n5);
    dollars := 0.5*n1 + 0.25*n2 + 0.1*n3 + 0.05*n4 + 0.01*n5;
    writeln;
    writeln('The piggy bank contains ',dollars:6:2,' dollars')
END.

```

## (e) PROGRAM sphere(input,output);

(\* This program calculates the volume and area of a sphere, given the radius \*)

```

CONST pi = 3.14159;
VAR radius,vol,area : real;
BEGIN
    writeln;
    writeln(' This program calculates the volume and area of a sphere. ');
    writeln;
    write(' What is the radius? ');
    readln(radius);
    (* Calculations.. *)
    area := 4.0*pi*sqr(radius);
    vol := area*radius/3.0;
    writeln;
    writeln(' The area is: ',area:8:2,' ,and the volume is: ',vol:7:2)
END.

```

```
(f) PROGRAM mass(input,output);

  (* This program calculates the mass of air in a tire *)

  VAR    P,V,m,T : real;

  BEGIN
    writeln;
    write(' This program calculates the mass of air in a tire');
    writeln(' using English units. ');
    writeln;
    write(' What is the Pressure, in psi? ');
    readln(P);
    writeln;
    write(' What is the Volume, in Cubic Feet? ');
    readln(V);
    writeln;
    write(' What is the Temperature, in degrees Fahrenheit? ');
    readln(T);

    (* The calculations *)

    m := (P*V)/(0.37*(T+460.0));

    (* Write out the answer *)

    writeln;
    writeln(' The Mass of air, in Pounds, is: ',m:7:2)
  END.
```

```
PROGRAM encode(input,output);

  (* This program will encode a five-letter word. *)

  VAR    a,b,c,d,e,a1,b1,c1,d1,e1 : char;

  BEGIN
    writeln(' This program will encode a five-letter word. ');
    writeln;
    write(' What word would you like encoded? ');
    readln(a,b,c,d,e);

    (* calculations *)

    a1 := chr(ord(a)-30);
    b1 := chr(ord(b)-30);
    c1 := chr(ord(c)-30);
    d1 := chr(ord(d)-30);
    e1 := chr(ord(e)-30);

    writeln;
    writeln(' The encoded word is: ',a1,b1,c1,d1,e1)
  END.
```

```
(h) PROGRAM decode(input,output);

  (* This program decodes a five-letter word. *)

  VAR    a,b,c,d,e,a1,b1,c1,d1,e1 : char;
```

**BEGIN**

```
writeln(' This program will decode a five-letter word. ');  
writeln;  
write(' What word would you like decoded? ');  
readln(a,b,c,d,e);
```

```
(* calculations *)
```

```
a1 := chr(ord(a)+30);  
b1 := chr(ord(b)+30);  
c1 := chr(ord(c)+30);  
d1 := chr(ord(d)+30);  
e1 := chr(ord(e)+30);
```

```
writeln;
```

```
writeln(' The decoded word is: ',a1,b1,c1,d1,e1)
```

**END.**

## الفصل السادس

- ٤٢ (a) - صحيح .  
 (b) - صحيح .  
 (c) - خطأ (أنواع بيانات مختلفة) .  
 (d) - خطأ .  
 (e) - صحيح .  
 (f) - خطأ (أنواع بيانات مختلفة) .  
 (g) - صحيح .  
 (h) - خطأ .  
 (i) - خطأ .
- ٤٣ (a) لا يمكن أن يظهر متغير من النوع الحقيقي في تعبير بولياني .  
 (b) - صحيح .  
 (c) لا يمكن استخدام متغير من النوع الحقيقي كمتغير تحكم .  
 (d) - صحيح .  
 (e) - صحيح على أن يكون  $b < c$  .  
 (f) القيم النسبية لكل من  $d$  ,  $b$  غير متناسقة .  
 (g) - صحيح .  
 (h) - إذا ما ظل flag صحيحا true ، فتستمر دورة REPEAT - UNTIL في التنفيذ لانهاثيا ، وإلا فإن دورة WHILE - DO تنفذ مرة واحدة فقط .  
 (i) - صحيح .  
 (j) - القائم بالاختيار لا يمكن أن يكون تعبيراً من النوع الحقيقي ، ولا يمكن للعناوين أن تكون حقيقية .  
 (k) لا يمكن نقل التحكم إلى عبارة مركبة .  
 (l) عناوين العبارات وعناوين الحالة لا يمكن استخدامها بالتبادل .

٥٠

```
PROGRAM loan(input,output);

(* This program generates a monthly payment schedule for a loan *)

VAR principal,rate,interest,monthly_interest,payment,ipart,itot,ppart : real;
    month : integer;

BEGIN

  (* Read the input data *)

  writeln(' This program generates a monthly payment schedule for a loan. ');
  writeln;
  writeln;
  write(' What is the amount of the loan? ');
  readln(principal);
  write(' What is the yearly interest rate, expressed as a percentage? ');
  readln(rate);
  interest := 0.01*rate;
  monthly_interest := interest/12.0;
  write(' What is the monthly payment? ');
  readln(payment);
```

(\* Carry out the calculations and write the results,  
on a month-by-month basis \*)

```

month := 1;
writeln(' Payment   Interest   Principal   Unpaid Balance   Total Interest');
WHILE principal*(1.0 + monthly_interest) >= payment DO
  BEGIN
    ipart := principal*monthly_interest;
    ppart := payment - ipart;
    principal := principal - ppart;
    itot := itot + ipart;
    write(' ',month:5,' ',ipart:8:2,' ',ppart:9:2,' ',principal:9:2);
    writeln(' ',itot:9:2);
    month := month + 1;
  END;
ipart := principal*monthly_interest;
ppart := ipart + principal;
principal := 0.0;
itot := itot + ipart;
write(' ',month:5,' ',ipart:8:2,' ',ppart:9:2,' ',principal:9:2);
writeln(' ',itot:9:2)
END.

```

(k) PROGRAM test(input,output);

(\* This program averages sets of student examination scores \*)

```

VAR avgtest,tottest,test : real;
    i,j,n,ntest,studid : integer;

```

BEGIN

```

  writeln(' This program averages n sets of student examination scores');
  writeln;

```

REPEAT

```

  write(' How many students are there? ');
  readln(n);
  IF n < 0 THEN writeln(' Illegal value, please try again..')
UNTIL n >= 0;

```

REPEAT

```

  write(' How many exams for each student? ');
  readln(ntest);
  IF ntest < 0 THEN writeln(' Illegal value, please try again..')
UNTIL ntest >= 0;
writeln;

```

FOR i := 1 TO n DO

```

BEGIN
    writeln(' Student no. ',i:2);
    writeln;
    write(' Enter the student's id: ');
    readln(studid);
    writeln;
    tottest := 0.0;
    writeln(' Enter the exam scores for this student');

    FOR j := 1 TO ntest DO
        BEGIN (* enter and sum the exam scores *)
            REPEAT
                write(' Test No. ',j:2,': ');
                readln(test);
                IF (test < 0.0) OR (test > 100.0) THEN
                    writeln('Illegal score, please try again..');
                UNTIL (test >= 0.0) AND (test <= 100.0);
                tottest := tottest + test
            END;

            (* calculate and write out the average *)
            avgtest := tottest/ntest;
            writeln;
            writeln(' Average score: ',avgtest:6:2);
            writeln
        END
    END.

(o) PROGRAM pyramid(output);

(* This program creates a pyramid of numbers in an interesting pattern *)

VAR      a,b,c,d,e,f,g,val : integer;

BEGIN
    FOR a := -9 TO 0 DO
        BEGIN
            b := (-1)*a;      (* Change counter to positive values *)
            c := a + 10;
            write('

                ');

            FOR d := 1 TO b DO write(' ');
            val := c + c - 1;

            FOR e := c TO val DO
                BEGIN (* Print left half of line, including center *)
                    IF (e >= 10) THEN f := e - 10 ELSE f := e;
                    write(f:1);
                END;
                val := c+c-2;
            FOR g := val DOWNT0 c DO
                BEGIN (* Print right half of line *)
                    IF (g >= 10) THEN f := g - 10 ELSE f := g;
                    write(f:1);
                END;
            writeln
        END
    END.
END.

```

```

(p) PROGRAM plot(input,output);

(* This program generates a plot of a damped sinusoidal function *)

CONST star = '*';
      blank = ' ';
      dash = '-';
      line = '|';
      linelength = 78;
VAR time,y,i,j,half : integer;

BEGIN
  writeln(' Generate a plot of the function exp(-0.1t) * sin(0.5t)');
  writeln;
  half := (linelength DIV 2);      (* Build a scale for the graph. *)

  FOR time := 0 TO 50 DO          (* Time 't' in seconds. *)
    BEGIN
      y := ROUND((exp((-0.1)*(time)))*(sin(0.5*time))*(half DIV 1)) + half;
      IF time = 0 THEN            (* Plot the vertical axis *)
        FOR i := 1 to linelength DO
          IF i = y THEN write(star)
          ELSE write(dash)
        ELSE
          (* Plot the function *)
          BEGIN
            IF half > y THEN j := half
            ELSE j := y;
            FOR i := 1 TO j DO
              IF i = y THEN write(star)
              ELSE IF i = half THEN write(line)
              ELSE write(blank)
            END;
          END;
        writeln
      END
    END.

```

## الفصل السابع

- ٣٥ (a) - لا يمكن أن يستخدم اسم الإجراء كمتغير  
 (b) - صحيح .  
 (c) - صحيح .  
 (d) - يحتوى هذا المثال على عدة أخطاء :  
 ١ - الدالة لا يوجد لها نوع بيانات .  
 ٢ - لا يمكن أن يظهر اسم الدالة فى توضيح VAR داخلى .  
 ٣ - المرفع الذى يمثل اسم الدالة لا يمكن أن يحدد له قيمة جديدة .  
 (e) - المؤشرات الفعلية من نوع مختلف عن المؤشرات الرسمية .  
 (f) - صحيح .  
 (g) - المؤشرات الفعلية المناظرة لمؤشرات المتغير الرسمية لا يمكن أن تكون تعبيرات ( لاحظ أن الإشارة إلى المتغير الشامل داخل الإجراء مسموح بها ) .  
 (h) - عدد المؤشرات الفعلية غير متساو مع عدد المؤشرات الرسمية المناظرة لها . كما أنه لا يمكن جمع المتغيرات من النوع الحرفى مع بعضها .  
 (i) - صحيح ( لاحظ أن الدالة تستخدم متغيرات شاملة ) .  
 (j) - صحيح .  
 (k) - يتغير المتغير الشامل C كل مرة يتم الاتصال فيها بالدالة ( تأثير جانبي side effect ) .  
 (l) - لا يمكن أن تمر الدوال القياسية كمؤشرات فى البسكال القياسى ( ISO ) .  
 (m) - صحيح .  
 (n) - يتم الاتصال بالدالة 1 funct ( داخل 1 proc ) قبل توضيحها .  
 (o) - يتم الاتصال بـ value ( داخل المجموعة الرئيسية ) خارج مدى توضيحها .  
 (p) - دالة الإعادة الذاتية لا تحتوى على شرط للإنتهاء .

٣٦

$$(a) \quad y = x_n + \sum_{i=1}^{n-1} x_i \quad \text{or} \quad y_n = x_n + y_{n-1}$$

$$(b) \quad y = \frac{(-1)^n x^n}{n!} + \sum_{i=0}^{n-1} \frac{(-1)^i x^i}{i!} \quad \text{or} \quad y_n = \frac{(-1)^n x^n}{n!} + y_{n-1}$$

$$(c) \quad p = f_i * \prod_{j=1}^{i-1} f_j \quad \text{or} \quad p_i = f_i * p_{i-1}$$

٤٢

```
PROGRAM powerfunction(input,output);
```

```
(* This program calculates the power of a number.
   The function is y = x to the nth power, where n is an integer. *)
```

```
VAR answer : char;
```

```
FUNCTION y(x : real; n : integer) : real;
```

```
(* this function raises x to the nth power *)
```

```
VAR product : real;
```

```
count : integer;
```



```

BEGIN
    (* form the product *)
    IF n = 0 THEN product := 1.0
    ELSE BEGIN
        product := x;
        IF abs(n) <> 1 THEN
            FOR count := 2 TO abs(n) DO
                product := product * x
            END;
    END;

    (* test for negative exponent *)
    IF n >= 0 THEN y := product ELSE y := 1.0/product
END;

PROCEDURE exponent;

(* this procedure evaluates the power function and writes out the result *)

VAR power,x : real;
    n : integer;

BEGIN
    writeln;
    write(' What is the base value? ');
    readln(x);
    write(' What is the value of the exponent? ');
    readln(n);
    power := y(x,n);
    writeln;
    writeln(' ',x:6:2,' Raised to the ',n:3,' power is ',power:10:4)
END;

BEGIN (* main action block *)
    REPEAT
        exponent;
        writeln;
        write(' Again (Y/N)? ');
        readln(answer)
    UNTIL ((answer = 'n') OR (answer = 'N'))
END.

```

٤٢

```

PROGRAM powerfunction(input,output);

(* This program calculates the power of a number.
   The function is  $y = x$  to the  $n$ th power. *)

VAR answer : char;

FUNCTION y(x,n : real) : real;

(* this function raises x to the nth power *)

BEGIN
    y := exp(n*(ln(x)))
END;

```

```

PROCEDURE exponent;

(* this procedure evaluates the power function and writes out the result *)

VAR power,x,n : real;

BEGIN
  REPEAT
    writeln;
    write(' What is the base value? ');
    readln(x);
    IF (x < 0.0) THEN (* write error message *)
      BEGIN
        writeln;
        writeln(' ERROR: The base value is negative. Try again');
        writeln
      END
    UNTIL x >= 0.0;
    write(' What is the value of the exponent? ');
    readln(n);
    power := y(x,n);
    writeln;
    writeln(' ',x:6:2,' Raised to the ',n:6:2,' power is ',power:10:4)
  END;
  BEGIN (* main action block *)
    REPEAT
      exponent;
      writeln;
      write(' Again (Y/N)? ');
      readln(answer)
    UNTIL ((answer = 'n') OR (answer = 'N'))
  END.

```

(g) PROGRAM test(input,output);

٤٩

```

(* This program averages sets of student examination scores *)

VAR avgtest,tottest : real;
    i,n,ntest : integer;

PROCEDURE readinput(n,ntest : integer; VAR tottest : real);

(* this procedure enters the input data for each student *)

VAR test : real;
    j,studid : integer;

BEGIN
  writeln(' Student no. ',i:2);
  writeln;
  write(' Enter the student's id: ');
  readln(studid);
  writeln;
  tottest := 0.0;
  writeln(' Enter the exam scores for this student');
  (* enter and sum the exam scores for each student *)

  FOR j := 1 TO ntest DO

```

```

BEGIN
    REPEAT
        write(' Test No. ',j:2,' : ');
        readln(test);
        IF (test < 0.0) OR (test > 100.0)
            THEN writeln('Illegal score, please try again..')
        UNTIL (test >= 0.0) AND (test <= 100.0);
        tottest := tottest + test
    END
END;

FUNCTION average(total : real; m : integer) : real;

(* this function determines an average from a total of m numbers *)

BEGIN
    average := total/m
END;

PROCEDURE writeanswer(avg : real);

(* this procedure writes out the average exam score for a particular student *)

BEGIN
    writeln;
    writeln(' Average score:   avg:6:2);
    writeln
END;

BEGIN (* main action block *)
    writeln(' This program averages n sets of student examination scores');
    writeln;

    REPEAT
        write(' How many students are there? ');
        readln(n);
        IF n < 0 THEN writeln(' Illegal value, please try again..')
    UNTIL n >= 0;

    REPEAT
        write(' How many exams for each student? ');
        readln(ntest);
        IF ntest < 0 THEN writeln(' Illegal value, please try again..')
    UNTIL ntest >= 0;
    writeln;

    FOR i := 1 TO n DO
        BEGIN
            readinput(n,ntest,tottest);
            avgtest := average(tottest,ntest);
            writeanswer(avgtest)
        END
    END.

```

```

(c) PROGRAM loan(input,output);

(* This program generates a monthly payment schedule for a loan *)

VAR principal,rate,interest,monthly_interest,amount : real;
    month,months : integer;

FUNCTION POWER(y : real; x : integer) : real;

(* this function raises y to the x power *)

BEGIN
    power := exp(x*(ln(y)))
END;

FUNCTION payment(principal,monthly_interest : real;
                 months : integer) : real;

(* this function determines the monthly payment *)

VAR quantity : real;

BEGIN
    quantity := power((1 + monthly_interest),months);
    payment := principal * monthly_interest * quantity / (quantity - 1)
END;

PROCEDURE readinput(VAR principal,monthly_interest : real;
                   VAR months : integer);

(* this procedure reads the input data *)

VAR rate,interest : real;
    years : integer;

BEGIN
    writeln;
    write(' What is the amount of the loan? ');
    readln(principal);
    write(' What is the yearly interest rate, expressed as a percentage? ');
    readln(rate);
    interest := 0.01*rate;
    monthly_interest := interest/12.0;
    write(' What is the duration of the loan, in years? ');
    readln(years);
    months := 12*years
END;

PROCEDURE writeoutput(principal,monthly_interest,amount : real;
                     months : integer);

(* this procedure generates and writes out a
   month-by-month payment schedule *)

VAR ipart,itot,ppart : real;
    month : integer;

```

```

BEGIN
write(' Payment   Interest   Principal   Unpaid Balance');
writeln('   Total Interest');
FOR month := 1 TO months DO
  BEGIN
    ipart := principal * monthly_interest;
    ppart := amount - ipart;
    principal := principal - ppart;
    itot := itot + ipart;
    write(' ',month:5,' ',ipart:8:2,' ',ppart:9:2);
    writeln(' ',principal:9:2,' ',itot:9:2)
  END
END;

BEGIN (* main action block *)
  writeln(' This program generates a monthly payment schedule for a loan. ');
  writeln;

  (* read the input data *)
  readinput(principal,monthly_interest,months);

  (* calculate the monthly payment *)
  amount := payment(principal,monthly_interest,months);
  write(' Monthly payment: ',amount:9:2);
  writeln;

  (* write the month-by-month payment schedule *)
  writeoutput(principal,monthly_interest,amount,months)
END.

```

### الفصل الثامن

. Miami - (a) ٢١

. Phoenix - (b)

. 6 - (c)

. 4 - (d)

. 4 - (e)

. صحيح - (f)

. خطأ - (g)

. (a) صحيح ٢٢

. (b) - التوضيحات موجودة في ترتيب خامس. وأنواع البيانات غير متناسقة في أول عبارة تحديد .

. (c) - البيانات المتعددة لا يمكن أن توجد في عبارة read ، أو عبارة write .

. (d) صحيح

. (e) - صحيح

```

PROGRAM dates(input,output);
(* This program determines the number of days a person has been alive *)

TYPE day = 1..31;
    month = 1..12;
    year = 1960..2100;
    numberofdays = 0..maxint;

VAR dd : day;
    mm : month;
    yy : year;
    birthday,today,days : numberofdays;
    n,a,m,e,s : char;

FUNCTION daysbeyond1960(mm : month; dd : day; yy : year) : numberofdays;
(* This function counts the days from January 1, 1960 to a given date *)

VAR n : numberofdays;
BEGIN
    n := trunc(30.42*(mm - 1)) + dd;
    IF mm = 2 THEN n := n + 1;
    IF (mm > 2) AND (mm < 8) THEN n := n - 1;
    IF (yy MOD 4 = 0) AND (mm > 2) THEN n := n + 1;
    IF (yy-1960) DIV 4 > 0 THEN n := n + 1461*((yy-1960) DIV 4);
    IF (yy-1960) MOD 4 > 0 THEN n := n + 365*((yy-1960) MOD 4) + 1;
    daysbeyond1960 := n
END;

PROCEDURE header(VAR n,a,m,e,s : char);
(* This procedure generates the conversational input *)

BEGIN
    write(' Please enter your first name (5 letters): ');
    readln(n,a,m,e,s);
    writeln;
    writeln(' Hello, ',n:1,a:1,m:1,e:1,s:1);
    write(' This program will calculate the number of days ');
    writeln(' you have been alive');
    writeln
END;

BEGIN (* main action block *)
    header(n,a,m,e,s);
    write(' When were you born, ',n:1,a:1,m:1,e:1,s:1,'? (mm dd yyyy) ');
    readln(mm,dd,yy);
    birthday := daysbeyond1960(mm,dd,yy);
    writeln;
    write(' What is today's date? (mm dd yyyy) ');
    readln(mm,dd,yy);
    today := daysbeyond1960(mm,dd,yy);
    days := today - birthday + 1;
    writeln;
    writeln(' ',n:1,a:1,m:1,e:1,s:1,', you have been alive ',days:5,' days.')
END.

```

```
PROGRAM dayoftheweek(input,output);
```

```
(* This program determines the day of the week for any given date *)
```

```
TYPE day = 1..31;  
      month = 1..12;  
      year = 1960..2100;  
      numberofdays = 0..maxint;
```

```
VAR dd : day;  
     mm : month;  
     yy : year;  
     n,a,m,e,s : char;
```

```
FUNCTION daysbeyond1960(mm : month; dd : day; yy : year) : numberofdays;
```

```
(* This function counts the days from January 1, 1960 to a given date *)
```

```
VAR n : numberofdays;
```

```
BEGIN
```

```
  n := trunc(30.42*(mm - 1)) + dd;  
  IF mm = 2 THEN n := n + 1;  
  IF (mm > 2) AND (mm < 8) THEN n := n - 1;  
  IF (yy MOD 4 = 0) AND (mm > 2) THEN n := n + 1;  
  IF (yy-1960) DIV 4 > 0 THEN n := n + 1461*((yy-1960) DIV 4);  
  IF (yy-1960) MOD 4 > 0 THEN n := n + 365*((yy-1960) MOD 4) + 1;  
  daysbeyond1960 := n
```

```
END;
```

```
PROCEDURE readinput(VAR mm : month; VAR dd : day;  
                   VAR yy : year; VAR n,a,m,e,s : char);
```

```
(* This procedure generates the conversational input *)
```

```
BEGIN
```

```
  write(' Please enter your first name (5 letters): ');  
  readln(n,a,m,e,s);  
  writeln;  
  writeln(' Hello, ',n:1,a:1,m:1,e:1,s:1);  
  writeln(' This program will calculate the day of the week');  
  writeln(' for any given date beyond January 1, 1960');  
  writeln;  
  write(' ',n:1,a:1,m:1,e:1,s:1,', please enter the date (mm dd yyyy) : ');  
  readln(mm,dd,yy)
```

```
END;
```

## الفصل التاسع

- ٤٤ (a) - صحيح .  
 (b) - صحيح .  
 (c) - صحيح .  
 (d) - الحدود الدنيا والعليا لأول فهرس موجودة في ترتيب خاطئ .  
 (e) - الفهرس لا يمكن أن يكون حقيقيا .  
 (f) - الفهرس لا يقع في المدى الصحيح .  
 (g) - صحيح .  
 (h) - مواصفات الفهرس في عبارة التحديد غير متناسقة مع توضيح المنظومة .  
 (i) - عنصر البيانات المحدد للعنصر المنظومة من نوع خاطئ .  
 (j) - الفهارس في عبارة التحديد من نوع بيانات خطأ .  
 (k) - صحيح .  
 (l) - لا يمكن أن تظهر محتويات المنظومة كمعناصر داخل تعبير عددي .  
 (m) - صحيح ( لأن name هو متغير سلسلة ) .  
 (n) - لا يمكن أن تظهر محتويات المنظومة في عبارات write , read ( وظهر متغير سلسلة في عبارة write هو استثناء ) .  
 (o) - صحيح .  
 (p) - تكوين خاطئ لعبارات unpack , pack ( فالفهارس مكتوبة بطريقة خاطئة ، كما أن قيسمها كبيرة جدا ) .  
 (q) - لا يمكن مقارنة سلاسل لها أطوال مختلفة ولا يمكن دمج متغيرات سلاسل ، أى لا يمكن إضافتها .  
 (r) - صحيح .
- ٤٥ (a) المؤشر الرسمى ( dummy ) يجب أن يكون من نفس نوع البيانات التى سبق تعريفها .  
 (b) - صحيح .  
 (c) - صحيح .  
 (d) - لا يمكن تحديد عنصر من عناصر منظومة مضمفوفة إلى إجراء أو إلى دالة كمؤشر فعلى .  
 (e) - صحيح .  
 (f) - المؤشر الفعلى الثانى ( cost ) من نوع مختلف عن المؤشر الرسمى المناظر له .  
 (g) - لا يمكن إلا ضغط آخر بعد فقط لمؤشر منظومة تكوينية متعددة الأبعاد .  
 (h) - يجب أن يكون المؤشر الرسمى في process مؤشر منظومة تكوينية متغيراً ، وليس مؤشر منظومة تكوينية قيمة .  
 (i) - المؤشرات الفعلية يجب أن تكون منظومات من النوع الحقيقى ، لها نفس الأبعاد .

## الفصل العاشر

- ٣٥ (a) - صحيح .  
 (b) - صحيح .  
 (c) - التعريف الصحيح لا يحتوى على END . كما أن تعريف نوع المنظومة ( sales ) غير صحيح أيضا .  
 (d) - صحيح .  
 (e) - تحتوى هذه المشكلة على العديد من الأخطاء :  
 ١ - لا يمكن أن تظهر محتويات السجلات في تعبير بوليان ( birthday = today ) .  
 ٢ - الإشارة إلى year في معيار التوقف ( UNTIL Year = 0 ) يجب أن يسبقها اسم السجل .  
 ٣ - القيمة النهائية المتوقعة في معيار التوقف ( Year = 0 ) تقع خارج المدى .



- (f) - صحيح .  
 (g) - يجب أن يتبع الجزء المتغير من السجل الجزء الثابت ( أى أن quantity : integer فى غير موضعها ) .  
 (h) - الإشارات إلى أسماء حقول السجل يجب أن يسبقها اسم السجل ( employees [i]. maritalstatus )  
 أو employees [i]. divorced . child ( كما أن عبارة writeln غير صحيحة ، حيث إن divorced ليس حقلا متغيرا نشيطا .  
 (i) - معرف حقل tag-field لا يمكن تمريره إلى إجراء كمؤشر متغير .

### الفصل الحادى عشر

- ٤٨ (a) - يجب أن تكون العبارة توضيحا متغيرا ، بدلا من تعريف للنوع .  
 (b) - صحيح .  
 (c) - يجب ألا تظهر FILE OF فى توضيح ملف النصوص .  
 (d) - صحيح .  
 (e) - مكون الملف لا يمكن أن يكون ملفا آخر .  
 (f) - التوضيح صحيح ، بالرغم من أنه غير ضرورى ، حيث إن input , output سبق توضيحهما كملفات نصوص .  
 (g) - الملف الذى سيكتب فى data لم توضع له قيم ابتدائية ( عن طريق rewrite ) . كما لا يمكن استخدام writeln أيضا ، حيث إن data ليس ملف نصوص .  
 (h) - صحيح .  
 (i) - صحيح ، على أن يوجد توافق ( customer . custname = name ) قبل ظهور نهاية الملف .  
 (j) - لا يمكن تمرير ملف إلى إجراء كمؤشر قيمة .

### الفصل الثانى عشر

- ٢٥ (a) [re,mi,fa,sol,la] (e) [do,re,mi,fa]  
 (b) [do,re,mi,fa,sol,la] (f) [ ]  
 (c) [do] (g) [ ]  
 (d) [re,mi,fa] (h) [do,re,mi]

- ٢٦ (a) [re,fa] (d) [do,re,mi,sol,ti]  
 (b) [do,re,fa] (e) [re,mi,fa,sol,la]  
 (c) [re,mi,fa,la,ti]

- ٢٧ (a) - خطأ .  
 (b) - صحيح .  
 (c) - خطأ .  
 (d) - صحيح .  
 (e) - خطأ .  
 (f) - صحيح .  
 (g) - صحيح .  
 (h) - صحيح .  
 (i) - خطأ .

(j) - صحيح .

(k) - صحيح .

(l) - خطأ .

(a) - صحيح .

٢٨

(b) - يجب أن يكتب تعبير بوليان على النحو التالي :

(c) - يجب أن تكتب عبارة التحديد على النحو التالي :

(d) - صحيح .

(e) - صحيح .

(f) - العنصر الأول في تعبير بوليان لا يمكن أن يكون متغيراً من نوع الفئة .

## الفصل الثالث مشر

```
(a) TYPE link = ↑customer;
      customer = RECORD
          name : PACKED ARRAY [1..20] OF char;
          acctno : 1..9999;
          balance : real;
          next : link
      END;
```

٣٥

(b) VAR nextcustomer, lastcustomer : link;

(c) new(nextcustomer);

.

.

.

dispose(nextcustomer);

٣٦

(a) lastcustomer := nextcustomer↑.next;

(b) nextcustomer↑.next := NIL;

(c) lastcustomer↑ := nextcustomer↑;

(d) IF nextcustomer↑.acctno = 1330 THEN  
nextcustomer↑.balance := 287.55;

OR

WITH nextcustomer↑ DO

IF acctno = 1330 THEN balance := 287.55;

(e) PROCEDURE readinput (VAR customer : link);

VAR count : 1..20;

BEGIN

.

.

.

count := 1;

WHILE NOT eoln DO

BEGIN

read(customer↑.name);

count := succ(count);

END;

```

.
.
END:

BEGIN (* main action block *)
.
.
    readinput(nextcustomer);
.
.
END.
( ) WHILE nextcustomer <> NIL DO
    BEGIN
        .
        .
        .
    END;

```

٣٧

١ - تخطيط البرنامج يحتوى على ٣ أخطاء :

- ١ - لا يتفق المعرف من نوع المشير مع اسم متغير الإشارة .
- ٢ - الحقل المشير داخل متغير الإشارة له تحديد نوع غير صحيح .
- ٣ - توضيحات المتغيرات لها تحديدات نوع غير صحيحة .

ب - تخطيط البرنامج يحتوى على خطأين :

- ١ - لا يحتوى تعريف متغير الإشارة على مشير .
- ٢ - المؤشر الذى يظهر فى عبارة new غير صحيح .

ج - صحيح . ( ينتج هذا البرنامج قائمة بالأسماء ) .





رقم الإيداع ١٩٩٠ / ٩٧٢٠

مطابع الكتب المصرية الحديثة  
MODERN EGYPTIAN PRESS  
مطابع الكتب المصرية الحديثة - ٢٢١١-٢٢١٢ - ٢٢١٣-٢٢١٤ - ٢٢١٥-٢٢١٦ - ٢٢١٧-٢٢١٨ - ٢٢١٩-٢٢٢٠ - ٢٢٢١-٢٢٢٢





## صدر أيضا للنشر

فى

## الحاسبات

- البرمجة بلغة الباسكال ..... كيلر
- الدوائر المتكاملة الرقمية والحاسبات ..... وولارد
- المجهزات والحاسبات الدقيقة لطلبة الهندسة والفنيين ..... وولارد
- الدوائر المنطقية واستخدامات المجهزات الدقيقة ..... موريس
- المدخل لعلم الحاسبات ..... باردتى
- البرمجة بلغة البيسك (شوم) ..... جوتفريد
- البرمجة بلغة الفورتران ..... لبيشتز
- الرياضيات الأساسية للحاسب (شوم) ..... لبيشتز
- كوك بييسك ..... جوتفريد
- موسوعة مصطلحات الكمبيوتر ..... محمود الشريف

## الناشر

### الدار الدولية للنشر والتوزيع

٢٨ ش الأهرام - روكسى - مصر الجديدة

ص. ب. ٥٥٩٩ هليوبوليس غرب - القاهرة

ت: ٢٥٨٢٨٨٧

تلكس: ٢٠٠٧٠ PBCRB UN

فاكس: ٢٩١٨٠٥٩ / ٠٠٢٠٢